Erweiterung des automatischen Testsystems AUSTERE

Sven Mertens

Konstanz, 28.02.2003

DIPLOMARBEIT

Diplomarbeit

zur Erlangung des akademischen Grades

Diplom-Informatiker (FH)

an der

Fachhochschule Konstanz

Hochschule für Technik, Wirtschaft und Gestaltung

Fachbereich Informatik/Wirtschaftsinformatik

Thema : **Erweiterung des automatischen Testsystems AUSTERE**

Diplomand : Sven Mertens, Eldoradostr. 8, 74211 Leingarten

Firma : Nokia GmbH, Lise-Meitner-Str. 10, 89081 Ulm

Betreuer : Detlev Albold und Helmut Malz

Eingereicht : Konstanz, 28.02.2003

INHALT

1 EINLEITUNG UND ÜBERBLICK

Zur größten Sparte der Produktfamilie der Firma Nokia gehört die Herstellung von Mobiltelephonen. Die Entwicklung, die Herstellung und das Testen der Produkte erfolgt an unterschiedlichen Standorten weltweit. Bei der Nokia GmbH in Ulm liegt der Schwerpunkt der Arbeit hauptsächlich beim Testen der Endgeräte auf Funktion und auf der Suche nach Fehlern bezüglich der Hardware als auch der Software. Zur Untersuchung der korrekten Funktionalität der Software werden unterschiedliche Testverfahren eingesetzt. Eine Einführung über verschiedene Verfahren und Methoden des Testens wie Regressionstests, Smoke-Tests, White- und Black-Box-Tests und weitere finden sich im Kapitel 2. Hier wird auf die allgemein bekannte Theorie von Testverfahren in Hinblick auf das verwendete automatische Testsystem eingegangen.

Zum automatischen Testen der Endgeräte bei Nokia wird ein computergestütztes System namens Austere eingesetzt. Dieses System wurde vollständig innerhalb der Firma konzipiert und wird ständig weiterentwickelt. Der erste Teil der Diplomarbeit bestand darin, ein solches Austere Testsystem aus neuen Hardwarekomponenten aufzubauen und die schon vorhandene Testsoftware namens RAPT darauf anzupassen. Der Aufbau sowie der Zusammenhang der Soft- und Hardwarekomponenten untereinander und die Handhabung des Systems wird in Kapitel 3 beschrieben. Erst wenn man die Komplexität des Systems und den Zusammenhang der Komponenten untereinander verstanden hat, kann man sich den Erweiterungen widmen. Sie gehören zum zweiten Teil der Diplomarbeit. Diese Erweiterungen werden in den Folgekapiteln behandelt. In ihnen wird für das jeweilige Thema zunächst die Theorie beschrieben und anschließend soll eine Lösung der Erweiterung des Austere-Systems gegeben werden.

Das Austere-System ist ein automatisches Testsystem, welches zum Testen der Mobilfunkgeräte Testskripte ausführt. Diese Skripte bestehen aus programmiertem Befehlscode, mit welchem das Austere-System beeinflusst oder konfiguriert werden kann und das Gerät, welches getestet werden soll, gesteuert und abgefragt werden kann. Hier setzt das größte Thema dieser Diplomarbeit an, nämlich Austere um neue Testverfahren zu erweitern oder anders ausgedrückt, die Skriptsprache um weitere Kommandos zu erweitern. Zu diesen Erweiterungen zählen: die Schaffung einer Grundlage, die integrierte Java Laufzeitumgebung (MIDP) abtesten zu können, die Möglichkeit Zeichen oder Schrift vom Bildschirm des Testgeräts auslesen zu können und Vergleiche von Tönen und/oder Musik einzuführen. Mit der Zeichen- oder Schrifterkennung öffnet sich ein weites Feld an Möglichkeiten. Man könnte z.B. in Abhängigkeit erkannter Zeichen das Testsystem beeinflussen und umkonfigurieren. Ein Vergleich von vorgegeben und erkannten Zeichen nach Art des schon vorhandenen Bildvergleichs lässt sich ebenfalls durchführen. Das letzte Thema, das des Audiovergleichs, beruht auf der Idee feststellen zu können, ob das Gerät eine Melodie oder einen Ton zu einem bestimmten Zeitpunkt ausgegeben hat oder nicht und ob diese Ausgabe mit einer zuvor aufgenommenen, einer Referenz, übereinstimmt.

Während der Arbeit an den genannten Erweiterungen kamen weitere Ideen, bereits vorhandene Befehle zu erweitern. Diese sind innerhalb dieser Ausarbeitung ebenfalls aufgeführt.

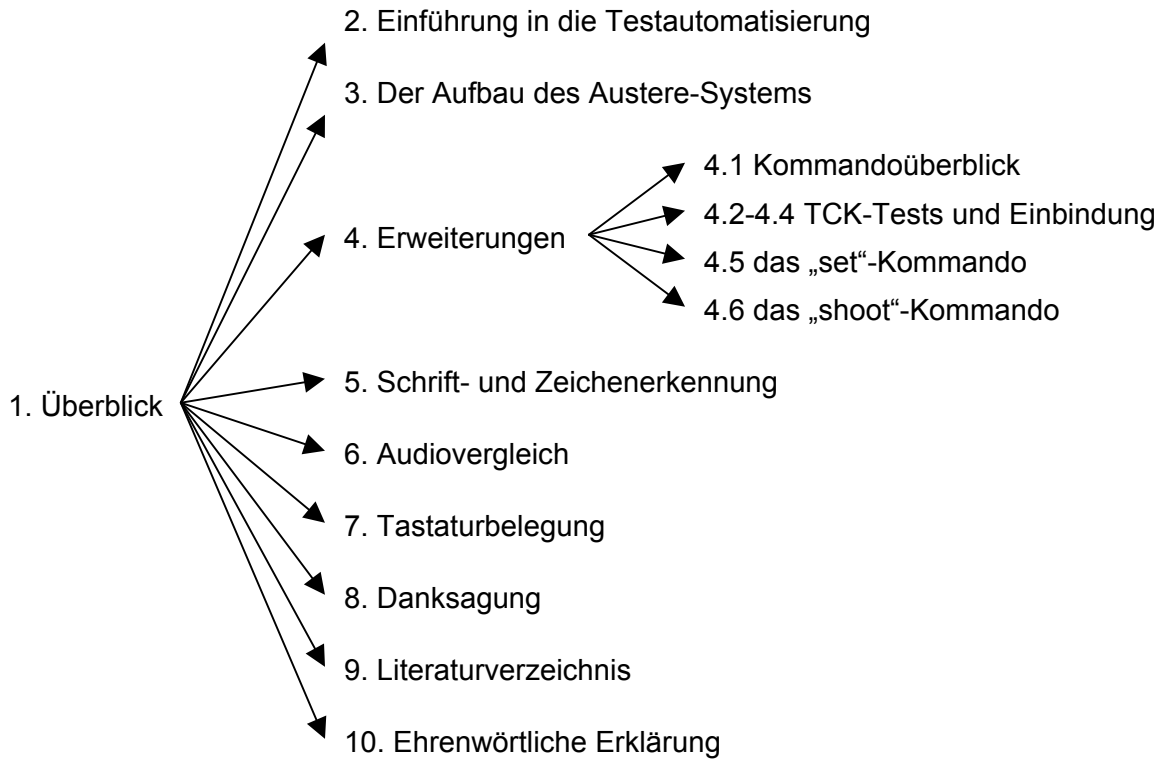
Ein Überblick der wichtigsten vorhandenen Befehle und die Erweiterungen neuer Befehle wird im Kapitel 4 behandelt. In diesem Kapitel wird ein Kurzüberblick über die Art und Weise des Testverfahrens der Java Laufzeitumgebung gegeben. Die Erweiterungen zur Einbindung dieser Testverfahren in das Austere-System werden im Anschluss daran erläutert. Zudem werden in diesem Kapitel neue und umstrukturierte Befehle, welche während dieser Diplomarbeit ausgearbeitet wurden, aufgeführt und beschrieben.

Den beiden größten Themen meiner Arbeit, nämlich der Zeichenerkennung und dem Audiovergleich widmen sich die Kapitel 5 und 6.

Im Kapitel 5 wird die Theorie neuronaler Netze beschrieben und eine Lösung der Problematik für die Schrift- und Zeichenerkennung gegeben. Die Anwendungsmöglichkeiten sowie die Analyse der Vorbedingungen und Einbindung innerhalb des Austere-Systems wird hier aufgezeigt.

Das Kapitel 6 beschäftigt sich mit dem Vergleich von Tönen oder ganzen Musikstücken. Eine Einführung in die Grundlagen der Mathematik von Fouriertransformation, Faltung und Korrelation ist hier gegeben, sowie die Analyse und Erörterung der Möglichkeiten und Beschränkungen sind hier aufgeführt. Zum Schluss wird eine Teillösung dieser Problematik und ihre Anwendungsmöglichkeiten aufgezeigt.

Hier noch einmal ein kleiner Überblick der Kapitel:



2 AUTOMATISIERUNG

2.1 Einführung in die Testautomatisierung

Die Zunahme der automatisierten Testfähigkeiten ist hauptsächlich durch die wachsende Zunahme der schnellen Anwendungsentwicklung bedingt, einer Entwicklungsmethode, die sich auf die Minimierung der Entwicklungszeit und die Bereitschaft häufiger, inkrementeller Software-Builds konzentriert. Die Testingenieure werden während der gesamten Entwurfs- und Entwicklungsphase an jedem Build beteiligt, um die Software zu verfeinern und dadurch sicherzustellen, dass sie den Bedürfnissen und Vorlieben des Benutzers genauer entspricht. In dieser Umgebung mit ständigen Änderungen und Ergänzungen der Software durch jeden einzelnen Build, in der die Entwicklung von Anforderungen gefördert wird, bekommt auch das Testen der Software einen iterativen Charakter. Jeder neue Build wird von einer beträchtlichen Anzahl neuer Tests sowie der Umarbeitung bestehender Testskripts durch Testingenieure begleitet, entsprechend der Umarbeitung bereits freigegebener Softwaremodule. Angesichts der ständigen Änderungen und Ergänzungen an den Anwendungen wird das automatisierte Testen der Software zu einem wichtigen Kontrollmechanismus, der die Korrektheit und Stabilität der Software in jedem Build gewährleistet.

Ein wesentliches Ziel besteht in der Verkürzung der Gesamtentwicklungszeit durch Angehen der risikoreichsten Entwicklungsaspekte in frühen Builds. Infolgedessen werden bereits zu Beginn des ersten Builds Testaktivitäten durchgeführt. Entwicklung und Entwurf von Tests stellen mitunter ein komplexes Unterfangen dar. Die Testarbeit kann genau so zeitaufwendig sein wie die Arbeit an der Entwicklung einer Anwendung.

Ein großer Teil des für ein Projekt benötigten Testaufwandes muss heute von automatisierten Testwerkzeugen unterstützt werden. Manuelles Testen ist arbeitsaufwendig und fehleranfällig, und es unterstützt nicht dieselben Qualitätsprüfungen, die mit einem automatisierten Testwerkzeug möglich sind. Ein solches Werkzeug kann sture manuelle Testaktivitäten durch eine effizientere und wiederholbare automatisierte Testumgebung ersetzen. Das Austere-System erweist hier seinen größten Wert, da Testskripte und Subroutinen von Testskripten wiederholt aufgerufen werden können. Jede neue Software oder Revision alter Software (nachfolgend Build genannt) bringt eine gewisse Anzahl neuer Tests mit sich, nutzt aber auch bereits entwickelte Testskripte. Dieses Testverfahren zahlt sich besonders aus, da diese Skripte wiederverwendbar sind und sie sehr häufig ausgeführt werden können. Sie lassen sich zudem leichter wiederholen als dies beim manuellen Testen möglich ist. Angesichts der ständigen Veränderungen und Ergänzungen an die Anforderungen und die Software dienen automatisierte Softwaretests als wichtiger Kontrollmechanismus zur Gewährleistung der Korrektheit und Stabilität der Software in allen neuen Builds.

Regressionstests auf Systemebene stellen ein weiteres Beispiel für die effiziente Verwendung automatisierter Tests dar. Sie sollen sicherstellen, dass die von einem System oder Programm bereitgestellten Funktionen wie festgelegt arbeiten und in der

Funktionsweise des Systems oder Programms keine unbeabsichtigten Änderungen aufgetreten sind. Ein Regressionstest ist ein Test oder eine Testreihe, der auf einem grundspezifizierten System oder Produkt ausgeführt wird, nachdem ein Teil der Systemproduktumgebung geändert wurde. Das Regressionstestpaket kann Testverfahren enthalten, die mit höchster Wahrscheinlichkeit die meisten Fehler entdecken. Diese Art des Testens sollte mit Hilfe eines automatisierten Werkzeugs durchgeführt werden, weil sie normalerweise langwierig und zäh ist, wenn sie manuell ausgeführt wird und deshalb anfällig für menschliche Fehler ist. Beim manuellen Testen kann es vorkommen, dass die beim ersten Durchgang unternommenen Schritte nicht genau dieselben sind wie beim zweiten Durchgang. Das Austere-System ermöglicht diese effiziente Durchführung von Regressionstests.

Smoke-Tests sind ein wenig umfangreicher und schneller als Regressionstests. Sie testen die wesentliche Funktionalität auf hoher Ebene. Regressionstests erweitern den Smoke-Test so, dass er die gesamte Funktionalität erfasst, die sich bereits als anwendbar erwiesen hatte.

Diese Tests zur Verifizierung von Software-Builds – auch Smoke-Tests genannt – konzentrieren sich auf die Automatisierung der Systemkomponenten, welche die wichtigste Funktionalität umfassen. Anstatt alles wiederholt neu zu testen, wenn ein neuer Software-Build herauskommt, führt der Testingenieur den Smoke-Test neu aus und überzeugt sich, dass die wesentliche Funktionalität des Systems noch gegeben ist. Er vergewissert sich, dass die Funktionalität, die in der vorigen Version des Programmcodes in Ordnung war, auch in der neuesten Version läuft. Smoke-Tests stellen zudem sicher, dass keine Arbeit durch Testen eines unvollständigen Builds vergeudet wird. Damit kann viel und wertvolle Zeit gespart werden. Das Ziel ist, sicherzustellen, dass die von dem modifizierten Produkt oder System bereitgestellten Funktionen die Spezifikationen erfüllen und keine wesentlichen Änderungen in den Betriebsfunktionen aufgetreten sind. Der gesamte Gedanke des Testens zielt darauf ab, Fehler zu finden, sie zu dokumentieren und sie schließlich zu beheben.

Das Austere-System kann ebenso die Smoke-Tests ausführen. Es führt wiederholt automatische Skripte aus, welche beim Auffinden von Fehlern genau die Folge von Befehlen wiederholen können, die zu dem Fehler geführt haben. So kann alles leicht reproduziert werden. Einzelne Funktionen, als auch das gesamte System kann damit abgetestet werden.

2.2 100% Tests

Jedoch sei hier auch mit der Vorstellung aufgeräumt, dass man mit einem solchen Testwerkzeug die Testarbeit zu 100% automatisieren könne. Es kann nicht alles getestet werden, und daher lässt sich auch nicht alles automatisieren. Gründe dafür sind durch die zeitliche und finanzielle Einschränkung gegeben. Die Zeit spielt eine Rolle bei der unendlichen Anzahl von Abwandlungen und Kombinationen von System- und Benutzeraktivitäten, die mit Benutzeroberflächen möglich sind. Diese Möglichkeiten ergeben sich aus den Pfaden, welche beschritten werden können, der Breite und Tiefe der Implementierung, aber auch aus möglichen Variationen der Eingabe von Benutzerdaten, seien sie gültiger oder ungültiger Art. Der Umfang an Abwandlungen und Kombinationen ist gewaltig. Automatisierung kann hier potenziell eine Aufgabe ohne Ende werden.

Automatisiertes Testen bringt jedoch auch Vorteile, wenn es korrekt implementiert ist und einem streng festgelegten Prozess folgt. Die Vorteile sind im wesentlichen die Erstellung eines zuverlässigen Systems, die Verringerung des Testaufwands und die Reduzierung des Zeitplans. Das beseitigt die Monotonie sich immer wiederholender Tests. Einfache sich

wiederholende Tests sind meist die Ursache dafür, dass viele Fehler nicht entdeckt werden. Angesichts des möglichen Umfangs jedes Tests muss sich der Testingenieur auf verschiedene Techniken wie die der Bildung von Äquivalenztests verlassen, die nur repräsentative Daten verwenden.

Das wesentliche Ziel besteht in der Gewährleistung, dass die Leistungsanforderungen an das System die Erwartungen der Benutzer erfüllen und / oder übertreffen.

2.3 Black-Box-Tests versus White-Box-Tests

2.3.1 White-Box-Tests

White-Box-Tests zielen auf die Erprobung innerer Aspekte des Zielprogramms ab. Sie versuchen logische Fehler zu finden. Bei diesem Testverfahren können Testpersonen den Programmcode einsehen und nach Fehlern suchen, die sich auf den Ausführungspfad, den Anwendungspfad, die Fallunterscheidungen und auf logische Konstrukte beziehen.

Diese Tests garantieren, dass

- alle Pfade in einem Modul mindestens einmal,
- Entscheidungsbäume nach allen Seiten,
- Schleifen an ihren Grenzen und innerhalb und
- Interne Datenstrukturen, auf ihre Gültigkeit

erprobt werden.

Diese Ebene des Testens untersucht den Steuerfluss (jeden Pfad), der auf der Ebene der Einheit ausgeführt wird, um nachzuweisen, dass der Programmcode korrekt ausgeführt wird, dass Algorithmen und Logik korrekt sind und die Einheit die Anforderungen und die ihr zugewiesenen Funktionen erfüllt. Fehlerbehandlungsroutinen müssen ebenso überprüft werden. Möglichkeiten bestehen hier in der Ersetzung einzelner Funktionen, die in sporadischen Fällen fehlerhafte Werte zurückgeben oder der Übergabe von ungültigen Werten an Funktionen. Solche Tests sollen Fälle aufdecken, in denen die Systemanwendung keine aussagekräftige Fehlermeldung zurückgibt, sondern abstürzt und einen Laufzeitfehler meldet. Sie stellen sicher, dass Fehler in der Zielanwendung gemeldet und korrekt behandelt werden. Diese Technik erkennt den Umstand an, dass es so gut wie unmöglich ist, auf jede denkbare Fehlerbedingung zu prüfen.

Zur Gattung der White-Box-Tests gehören ebenso die Einheitstests, welche die Übereinstimmung des Codes mit dem Design, die Pfade durch den Code, die richtige Formatierung von Bildschirmen, Meldungen, Bereich und Typ von Eingaben und die Ausgabe geeigneter Ausnahmen oder Fehlermeldungen überprüfen. Zu den Fehlern können Logikfehler, Fehler durch Überlastung oder durch Bereichsüberschreitung, Zeitfehler und Fehler durch Speicherprobleme gehören. Zu letzteren Fehlern können Spezialwerkzeuge eingesetzt werden, welche über mehrere Stunden oder Tage die Speichernutzung der Anwendung protokollieren. Sie stellen fest, ob der Speicherverbrauch weiter steigt und sind

sogar in der Lage, die Anweisungen zu ermitteln, bei denen zugewiesener Speicher nicht wieder freigegeben wird.

Als letztes seien hier noch die Integrationstests und Modultests aufgeführt. Die Integrationstests haben den Zweck der Prüfung, ob jede Softwareeinheit korrekt mit anderen Einheiten zusammenarbeitet. Diese Technik untersucht nicht nur die übergebenen Parameter von einer Einheit zur nächsten, sondern auch die globalen Parameter sowie bei objektorientierten Anwendungen alle Klassen. Modultests ermitteln, ob die Module erfolgreich eine zusammenhängende Einheit bilden und ob diese präzise und konsistente Ergebnisse liefert. Hier wird jede Funktion des Moduls auf Fehler und Vollständigkeit untersucht; sämtliche Anweisungen und bedingte Schleifen werden hier mit eingeschlossen.

2.3.2 Black-Box-Tests

Demgegenüber stehen die Verfahren der Black-Box-Tests. Sie überprüfen das externe Verhalten von Eingaben und zugehörige Ausgaben und die Funktionalität der Software entsprechend den Anforderungen. Diese Tests verwenden nur etablierte öffentlich zugängliche Schnittstellen wie die Benutzerschnittstelle oder die öffentliche Schnittstelle für Anwendungsprogrammierung (API). Sie erproben die Systemanwendung im Hinblick auf Funktionsanforderungen mit der Absicht, eine etwaige Nichtübereinstimmung mit den Anforderungen des Endbenutzers aufzudecken. Das wichtigste Ziel besteht darin festzustellen, ob mit der Anwendung die Aufgaben erfüllt werden können, für die sie erstellt wurde.

Das Austere-System basiert alleinig auf dem Black-Box-Testverfahren. Dem Testingenieur des Systems fehlt bewusst das Wissen um interne Strukturen und Aufbau. Denn diese Art des Testens versucht vielmehr aus Sichtweise eines Benutzers

- inkorrekte oder fehlende Funktionalität,
- Schnittstellenfehler,
- Probleme beim Sichern und Wiederherstellen und
- Sicherheitsprobleme

zu finden.

Die Black-Box-Tests lassen sich in Techniken verschiedener Art unterteilen, welche nachfolgend kurz beschrieben werden.

Eine Testtechnik macht Gebrauch von der Bildung von Äquivalenzklassen. Diese Technik der Klassenbildung hat den Vorteil der Verringerung des Testumfangs. Wie schon erwähnt, ist erschöpfendes Testen aller möglichen Eingaben nicht möglich, deshalb müssen die Tests mit einer Teilmenge der Eingaben durchgeführt werden. Zum Beispiel werden viele Eingabemodule im Code wiederverwendet. Hier reicht es meist aus, mit jeweils einer Art von Eingabefeld alle Grenzfälle abzudecken. Zum Beispiel werden beim Testen von Grenzfällen Werte, die innerhalb, auf der Grenze und über die Grenze hinausgehen, sowie Höchst- und Tiefstwerte verwendet. Ebenso werden die Felder auf gültige und ungültige Eingabewerte hin getestet. Zusätzlich wird überprüft, ob numerische, alphabetisch oder alphanumerische Werte vom Feld akzeptiert werden. Die Testergebnisse sind dann repräsentativ für alle Eingabefelder gleicher Art im gesamten System.

Testingenieure, die sich hauptsächlich nur mit den Black-Box-Techniken beschäftigen, verwenden auch häufig der Begriff Systemtest dafür. Die Systemtests umfassen Unterarten wie Funktions-, Datenintegritäts-, Sicherheits-, Belastungs-, Leistungs- und Betriebsbereitschaftstests.

Die **Funktionstests** beruhen auf Testverfahren, welche die Funktionalität des Systems auf der Grundlage der Projektanforderungen erproben. Sie erproben die Systemanwendung auf Funktion, wobei versucht wird, etwaige Nichtübereinstimmungen mit den Anforderungen des Endbenutzers aufzudecken. Dazu sollten die Tests in Gruppen organisiert werden, um doppelte Arbeit zu vermeiden. Diese Tests lassen sich dann in bestimmter Reihenfolge ausführen und gegebenenfalls mehrfach wiederverwenden und gemeinsam nutzen.

Zum **Sicherheitstests** gehört die Prüfung der korrekten Funktion von System- und Datenzugriffsmechanismen. Zur Prüfung von Sicherheitsstufen und Zugriffsbeschränkungen und damit zur Sicherstellung der Erfüllung der festgelegten Sicherheitsanforderungen und Sicherheitsregelungen werden diese Tests ebenso verwendet.

Die **Belastungstests** erproben das System ohne Rücksicht auf das Design. Diese Tests sollen Fehler finden, welche durch diese Beschränkungen hervorgerufen werden. Zudem werden Kapazität und Elastizität des Systems gemessen. Das System muss große Datenmengen verarbeiten und innerhalb kürzester Zeit viele Funktionsaufrufe ausführen. Eine weitere Möglichkeit besteht in Kombination bei gleichzeitiger Durchführung anderer Tests.

Verschiedene Unterarten der **Belastungstests** sind die Einheitsbelastungstests, welche Belastungen für eine einzelne Komponente erzeugen, die Modulbelastungstests, in welchen Funktionen in einem gemeinsamen Bereich verarbeitet werden und die Systembelastungstests, welche das System mit einem hohen Transaktionsvolumen belasten. Zu letzterem gehören auch die Parallelitätstests, welche verifizieren sollen, dass die Anwendung oder das System mit dem Zugriff auf mehrere Module und Daten zurechtkommt. Bei den Belastungstests wird gefragt, was geschieht, wenn das System bis an die Grenzen und darüber hinaus getrieben wird.

3 DER AUFBAU VON AUSTERE

3.1 Austere Übersicht

Unter dem Namen Austere ist ein Komplettsystem mit folgenden Untersystemen abgedeckt:

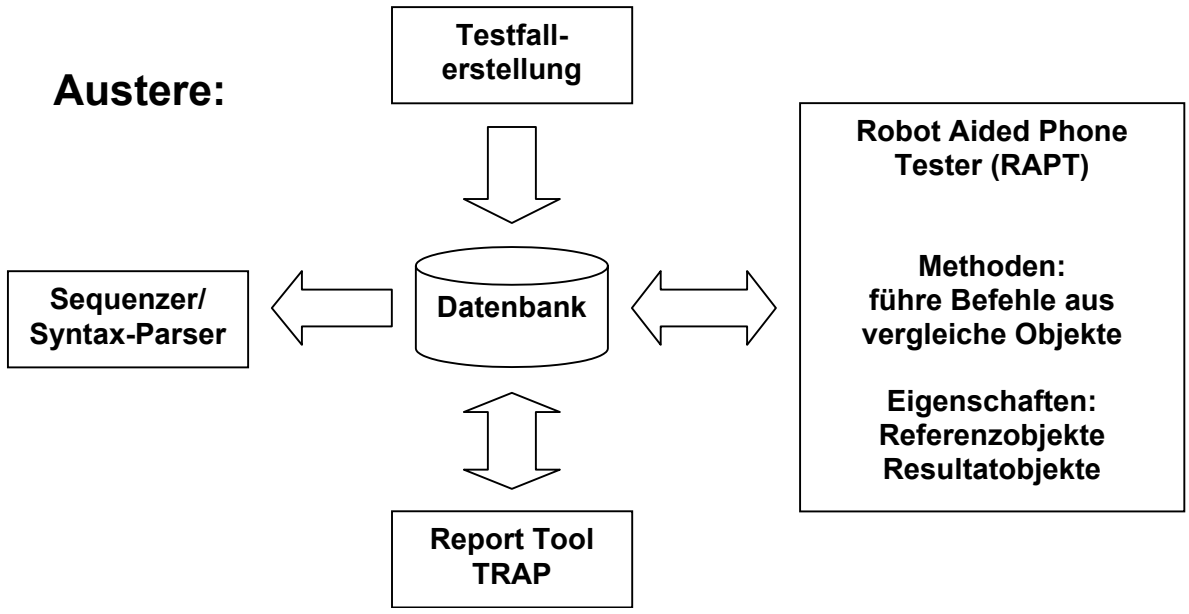


Abbildung 3.1: Austere im Überblick

Austere ist ein automatisches Testsystem für Mobilfunkgeräte bei Nokia Mobile Phones. Dieses System wird verwendet, um neue Software-Builds auf Basis der Benutzeroberfläche (UI) zu testen. Das Testsystem simuliert das Drücken der Tasten des zu testenden Mobiltelephons über ein angeschlossenes Schaltersystem, und über ein Kamerasystem kann die Bildschirmausgabe im Anschluss aufgenommen und überprüft werden. Der Sequenzer oder Syntax-Parser, die Testfallerstellung, das Report Tool und der Robot Aided Phone Tester sind unabhängige Bestandteile des Austere-Systems, welche allesamt die Datenbank als gemeinsame Schnittstelle zueinander benutzen.

Mit Hilfe der Testfallerstellung lassen sich Testfälle bzw. Testskripte erstellen und in der Datenbank ablegen. Für weitergehende Erklärung siehe Kapitel 3.2.

In der Datenbank liegen die Testskripte für das Testsystem vor. Aus dieser werden sie vor der Ausführung geholt und anschließend ausgeführt. Die Resultate der Tests werden

abschließend zurück in die Datenbank überführt. Eine Grobskizzierung der Datenbank befindet sich in Kapitel 3.3.

Das Report Tool (TRAP) ermöglicht die Validierung und Verifizierung abgeschlossener Tests und ganzer Testreihen. Eine weiterführende Erklärung hierzu findet sich im Kapitel 3.4.

Der Sequenzer oder Syntax-Parser ist ein kleines Programm innerhalb der Oberfläche von Access (siehe Abbildung 3.3). Nach einer Testfallerstellung kann mittels diesem Hilfsprogramm die Syntax der erstellten Testskripte überprüft werden bevor es zum eigentlichen Testen kommt.

Der Robot Aided Phone Tester ist letztlich das Hauptprogramm. Hierin werden die Testfälle bzw. Befehle ausgeführt, Referenzobjekte aus der Datenbank geholt, mit dem Resultatobjekt des jeweiligen Tests verglichen und die Resultate werden zurück in die Datenbank geschrieben. Der Aufbau von RAPT wird im Kapitel 3.5 näher durchleuchtet.

3.2 Design von Testfällen

Das Entwerfen des Testprogramms folgt auf die Testanalyse, zu deren Ergebnissen die Definition von Testzielen sowie die Erstellung von Testanforderungsbeschreibungen gehört. Das Modell des Testprogramms besteht aus einer grafischen Darstellung, die den Anwendungsbereich des Testprogramms veranschaulicht. In Abbildung 3.2 ist ein Ausschnitt aus dem Modell dargestellt. Er zeigt einen Teil der Menüführung innerhalb des Menüs „Profile“.

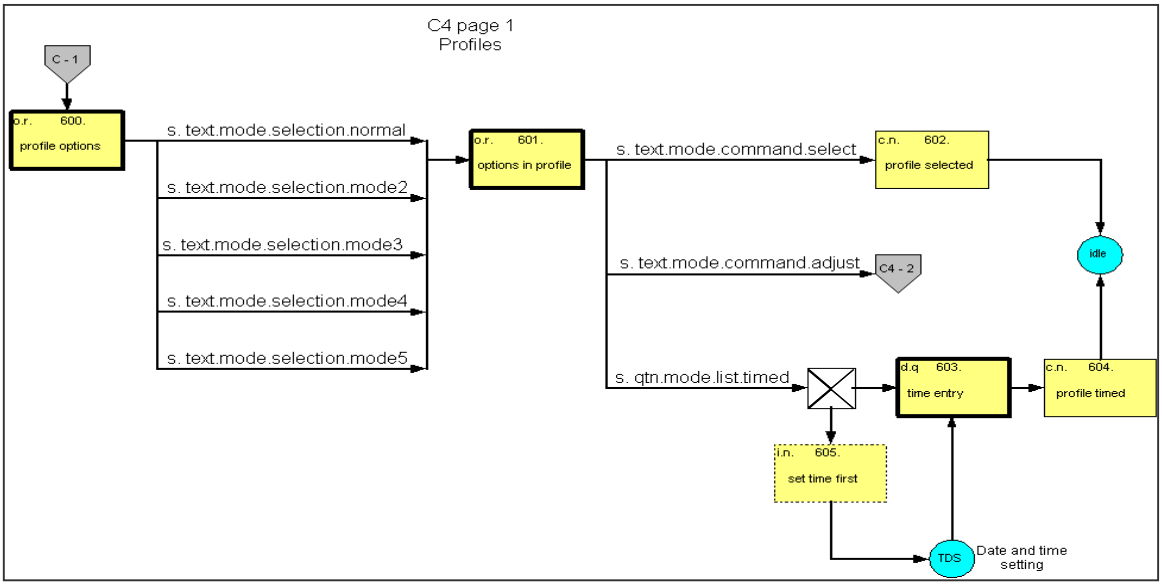


Abbildung 3.2: Teilablaufdiagram innerhalb des Hauptmenüs Profile

Durch die Auswahl eines Profils innerhalb des Profilmenüs (600) gelangt der Benutzer in die Optionen (601) desselbigen. Hier kann das Profil aktiviert (select), angepasst (adjust) oder es können Zeiteinstellungen vorgenommen werden.

Diese Testarchitektur liefert dem Testteam eine Wegbeschreibung zur Ermittlung der verschiedenen für die einzelnen Designkomponenten erforderlichen Testskripte. Die Designkomponenten sind in Form ihrer Design-IDs aufgeführt: Die ID 600 steht hier für die Testgruppe 600. Das Design umfasst die Gliederung der Testverfahren in logische Gruppen und die Definition einer Namenskonvention für das Verfahrenspaket, später die Einteilung in Testgruppen.

Dieses Modell enthält die Testtechniken, die auf Entwicklungs- und Systemtestebene eingesetzt werden. Es skizziert statische Teststrategien, welche u.a. während der Anwendungsentwicklung genutzt werden. Dieses Modell ist zugleich Struktur des Testprogramms, deren Ziel die Definition der Methode für die Organisation der Testverfahren ist. Die Testverfahren befassen sich mit den Vorbedingungen für einen Test, den erforderlichen Dateneingaben, den nötigen Aktionen, den erwarteten Ergebnissen und den Verifizierungsmethoden. Da das Ziel der Testarbeit darin besteht, Fehler in der zu testenden Anwendung aufzudecken und gleichzeitig die Erfüllung der Testanforderungen zu verifizieren, besteht ein wirkungsvolles Testverfahrendesign aus Tests, die mit hoher Wahrscheinlichkeit bisher nicht entdeckte Fehler finden. Ein gutes Testverfahrendesign deckt nicht nur Ein- und Ausgaben ab, sondern versucht auch, unerwartete Werte für Ein- und Ausgaben zu berücksichtigen.

Diese designorientierte Testarchitektur verknüpft der Testingenieur mit den Testverfahren, mit womit Hard- und Softwaredesignkomponenten der Systemanwendung getestet werden. Die Testanforderungsbeschreibungen lassen sich bis zu den Designkomponenten sowie zu den System- und Softwareanforderungsspezifikationen verfolgen.

Das Werkzeug zur Erstellung der Testskripte besteht aus einer Sammlung von Formularen für die Eingabe der Kommandos zur Ausführung durch das Austere-System, den Testfallbeschreibungen und weiteren mehrheitlich textuellen Informationen. Die Design-IDs spiegeln sich aber auch in den Spezifikations-IDs (Spec-ID) wieder, unter welchen die

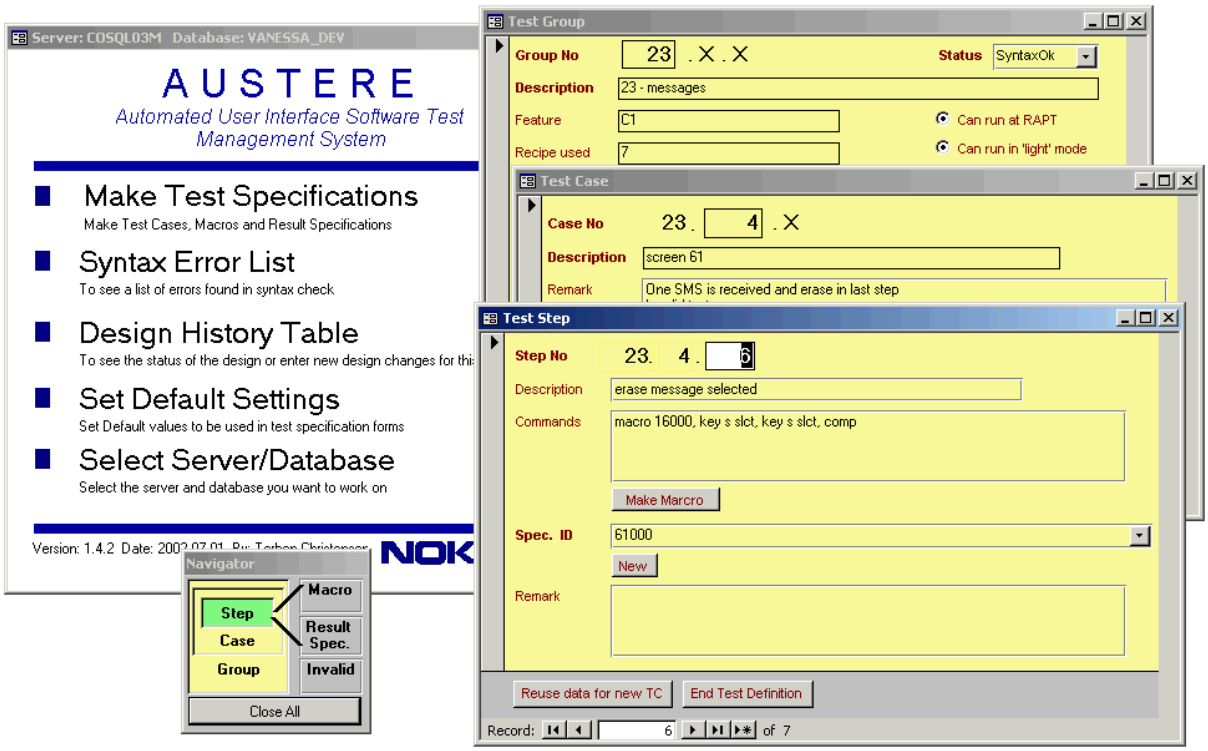


Abbildung 3.3: Eingabemasken zur Erstellung der Testfälle aus dem Menü „Make Test Specifications“; Realisierung durch das Programm MS-Access.

Referenzen abgelegt werden. Referenzen können einzelne Schnappschüsse des Bildschirms (Bitmap), eine Sequenz aus mehreren Schnappschüssen (Video) oder aber aufgenommene Töne oder Musik (Wave) sein. Sie werden als Objekt in der Datenbank abgelegt.

Die Testspezifikationen sind unterteilt in Gruppen: den Testgruppen (auch als Test Groups oder TGroup bezeichnet), den Testfällen (auch Test Cases oder TCase) und den Testschritten (Test Steps oder TSteps). Eine Testgruppe kann als Container für eine bestimme Menge an Testfällen betrachtet werden. Innerhalb einer Gruppe werden meist Tests zusammengefasst, welche eine bestimmte Grundeinstellung und/oder Konfiguration benötigen oder voraussetzen. Eine Gruppe beinhaltet beispielsweise nur Testskripte für die Nachrichtendienste, eine andere ist nur für Profildienste.

Ein Testfall kann wiederum als Container für untergeordnete Testschritte angesehen werden. Vor der Ausführung untergeordneter Testschritte kann eine Vorbedingung programmiert werden, welche eine Initialisierung oder Änderung der Grundeinstellung vornimmt.

Die eigentliche Testausführung beruht letztlich auf den in den Testschritten ausprogrammierten Kommandos. Das am häufigsten gebräuchliche Kommando ist das „key“-Kommando zur Steuerung des Schaltsystems und damit der Simulation von Tastendrücken, des weiteren das „comp“- und „shoot“-Kommando zur Aufnahme eines Schnappschusses mit bzw. ohne nachfolgenden Vergleich mit der Referenz und das „macro“-Kommando, welches vereinfachte Befehle innerhalb eines Testskripts zulässt.

3.3 Die Datenbank

Auf einem zentralen SQL-Server befinden sich alle Datenbanken für die Austere-Systeme. Diese Systeme unterscheiden sich hinsichtlich der zu testenden Geräte, der verwendeten Kameras für die Displayaufnahmen und den Zusatzmodulen von RAPT selbst. Die verschiedenen Datenbanken unterscheiden sich hinsichtlich unterschiedlicher zu testender Geräte, des verwendeten Austere-Systems und der zu erledigenden Testreihen. Das in Abbildung 3.3 zu sehende des MS-Access Frontend zur Erstellung der Testskripte kann individuell mit der benötigten Datenbank verlinkt und genutzt werden. Die Transaktionen von und zum SQL-Server laufen über einen geöffneten TCP/IP-Port des Servers.

Die Tabellen einer solchen Datenbank sind in vier Gruppen nach ihren Funktionalitäten unterteilt. Zur ersten Gruppe gehören die Definitionen der Testfälle. Hierin stehen die IDs, die Beschreibungen der Skripte und erwarteten Ergebnisse und zu erfüllende Voraussetzungen für die Ausführung. Diese Gruppe setzt sich im wesentlichen aus den Tabellen Macro, TGroup, TCase und TStep zusammen. In der Tabelle Macro werden Makros gehalten, welche vereinfachte und wiederverwendbare Befehlsstrukturen für die Testskripte erlauben. Die wichtigsten Einträge eines Makros sind: die Macro-ID, die Kommandos und ihre Beschreibung. Makros können bis zu einer Tiefe von vier wieder andere Makros beinhalten. In den Tabellen TGroup, TCase und TStep liegt die Struktur der Ausführung vor. Die wichtigsten Einträge hier sind: die Gruppen-, Fall-, und Schritt-IDs, die Beschreibungen und die Skriptkommandos (zur

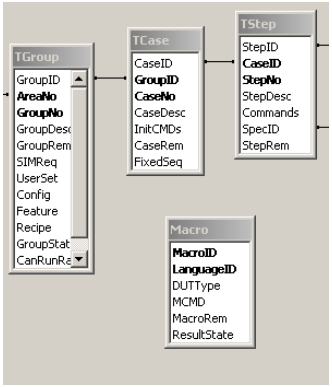


Abb. 3.4: Tabellen für Testfälle

Initialisierung nachfolgender Testschritte in der Tabelle TCase und Ausführungskommandos in der Tabelle TStep. Jede Ausführung eines Testschrittes muss mit einem abschließenden Vergleich enden.

Zur zweiten Gruppe gehören alle Tabellen, welche mit Daten vom Testsystem gefüllt werden. Die wichtigsten sind die Tabellen Result und ResultAsObj. In die Tabelle Result werden vom Austere-System Daten wie die ID des Testschrittes (StepID), das Ausführungsresultat in Form einer Zahl und eines Textes sowie der Zeitstempel eingetragen. In die Tabelle ResultAsObj werden bei Nichterfolg die Resultatobjekte und das zugehörige Format eingetragen. Anmerkung: ein Resultat wird dann als nicht erfolgreich gewertet (failed oder manual), wenn der Vergleich von Referenz zu Resultat negativ ausgefallen ist oder durch das Kommando „shoot“ ein Schnappschuss von der Benutzeroberfläche genommen wurde.

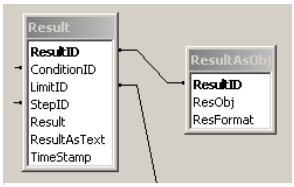


Abb. 3.5: Tabellen für Resultate

Die dritte Gruppe beinhaltet alleinig die Tabelle RefObject, in welcher die Referenzobjekte der Tests gehalten werden. Diese Objekte werden beim ersten Durchlauf eines Testskripts aufgenommen. Abschließend müssen sie manuell über das Report Tool TRAP bestätigt werden.

Zur vierten Gruppe schließlich gehören Tabellen, welche zur Unterstützung und Konfiguration des Testsystems dienen, dies sind die sogenannten Hilfstabellen. Eine Tabelle ist TLanguage, in welcher die Sprachunterstützung des zu testenden Gerätes konfiguriert werden kann. Eine andere ist die History-Tabelle, welche Aufschluss über Änderungen gibt.

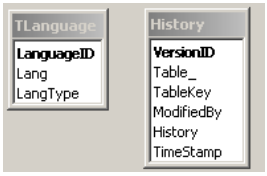


Abb. 3.6: Hilfs-tabellen

Die Datenbank beinhaltet sozusagen alle Testfälle oder auch Kommandos für das Testsystem, die erwarteten Testergebnisse (Referenzobjekte) und die aktuellen Testergebnisse aus laufenden und/oder abgeschlossenen Tests. Die Objekte, ob Bilder im Bitmap-Format, Töne im Wave-Format oder Videos im AVI-Format, werden in ihrem originalen Dateiformat in der Datenbank abgelegt.

3.4 Das Report-Tool TRAP oder die Ergebnisanalyse

TRAP bedeutet Test Results Automatically Presented und wird vom Standort Kopenhagen betreut und weiterentwickelt. TRAP unterstützt zum jetzigen Zeitpunkt etwa 25 verschiedene Systeme, darunter alle Austere-Systeme. Zwei der wichtigsten Menüpunkte dieser Software sind die „Approval Form“ und „Change Results“.

Das Approval Formular (Abbildung 3.7) dient der Bestätigung neu aufgenommener Referenzen durch das Testsystem. Auf der rechten Seite zu sehen ist das neue Referenzbild, welches zu untersuchen gilt. Auf der linken Seite stehen Informationen über den ausgeführten Testfall. In der Mitte hat der Testingenieur die Möglichkeit sich über Annahme, Verwurf oder noch ausstehende neue Referenz zu entscheiden. Über einen Filter kann der Tester im Vorfeld eine Auswahl über bestimmte Testgruppen und Testreihen vornehmen und somit die Menge der Anzeigen reduzieren.

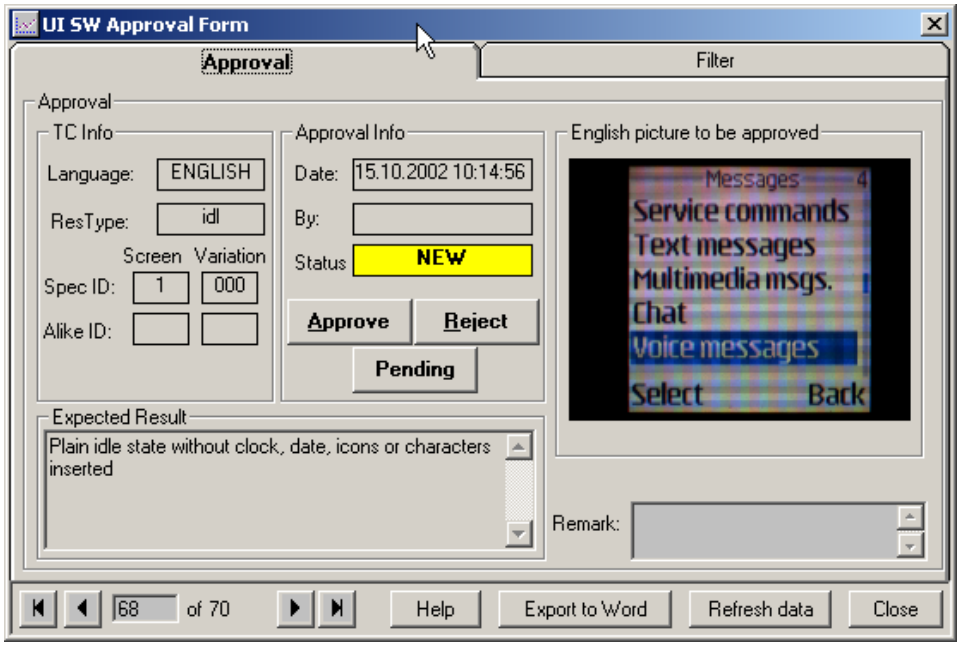


Abbildung 3.7: Approval Form – Formular zur Verifizierung der ersten Resultate

Abbildung 3.8: Change Results – Formular zur Gegenüberstellung von Referenz und Resultat

Das Change Results Formular in Abbildung 3.8 stellt Referenz (rechts) und Resultat (links) mit entsprechenden Zusatzinformationen gegenüber. In diesem Formular können Referenz- und Resultatbilder direkt gegenübergestellt werden. Somit ist ein direkter Vergleich gegeben. In einer der letzteren Versionen wurde es um die Fähigkeit der Gegenüberstellung von Tönen und Videosequenzen erweitert. Bei Tönen erscheint stellvertretend ein Lautsprechersymbol, welches auf Abruf die Aufnahmen abspielen kann.

Dem Benutzer gibt dieses Formular die Möglichkeit das Resultat zu bestätigen oder letztlich als fehlerhaft zu deklarieren. Ab und an kann es vorkommen, dass vermeintliche Fehler gar keine sind; dies kann vorkommen, wenn Tests viele Transaktionen vornehmen und die Aufnahmen der Ergebnisse unterschiedlich ausfallen können. Ein vermeintlich negatives Resultat aus einem Vergleich heraus kann im nachhinein positiv bewertet und als neue Referenz übernommen werden. Dieses Tool ermöglicht sozusagen die Auswertung einer Testreihe.

Maßnahmen zum Berichten eines Fehlers sind zudem in TRAP gegeben. Hier können fehlerhafte Vergleiche in Dateien für andere Programme wie MS-Word oder –Excel exportiert werden. Diese Ergebnisse können so noch einmal überflogen und anschließend an das Anwendungsentwicklungsteam weitergeleitet werden. Eine Zuordnung nach Prioritäten für die Fehler kann ebenso stattfinden.

TRAP führt somit den Beweis, dass das aufgebaute System allen angegebenen Anforderungen entspricht. Da jede Stufe der Systementwicklung die Anforderungen formt, partitioniert und organisiert, um sie näher an die Form des neuen Systems zu bringen, sind die Testingenieure mit Hilfe von TRAP in der Lage, die ursprünglichen Anforderungen zu Testzwecken auf die Lösung abzubilden.

3.5 Der Robot Aided Phone Tester

3.5.1 Übersicht der verwendeten Komponenten

Das komplette RAPT-System wurde aufgebaut aus folgenden Hard- und Softwarekomponenten:

- der Agilent E3632A Spannungsversorgung (1),
- dem Agilent 34970A Messdatenerfassungs- und Schaltsystem (2),
- der National Instrument PCI-GPIB Karte zur Ansteuerung der Spannungsversorgung und des Schaltsystems sowie ggf. einem GSM-Netzwerksimulator,
- der Sony XC-003P Kamera (3),
- der Cognex 8120 Frame Grabber Karte zur Ansteuerung der Kamera,
- dem Cognex Programm DisplayInspect (4),
- der Agilent Programmierumgebung VEE pro (4),
- und einem etwas modifizierten Mobiltelephon (5).

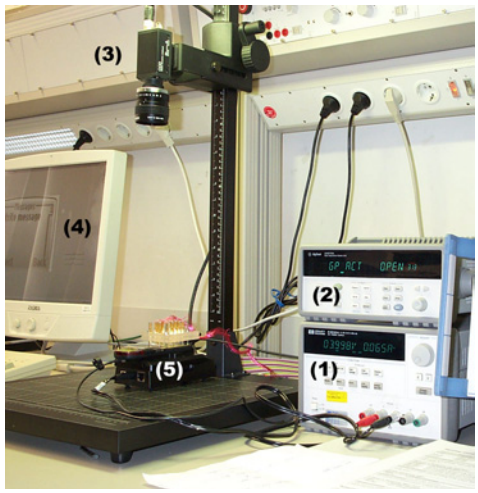


Abbildung 3.9: beispielhafter technischer Aufbau

3.5.2 Der Aufbau der Hardware

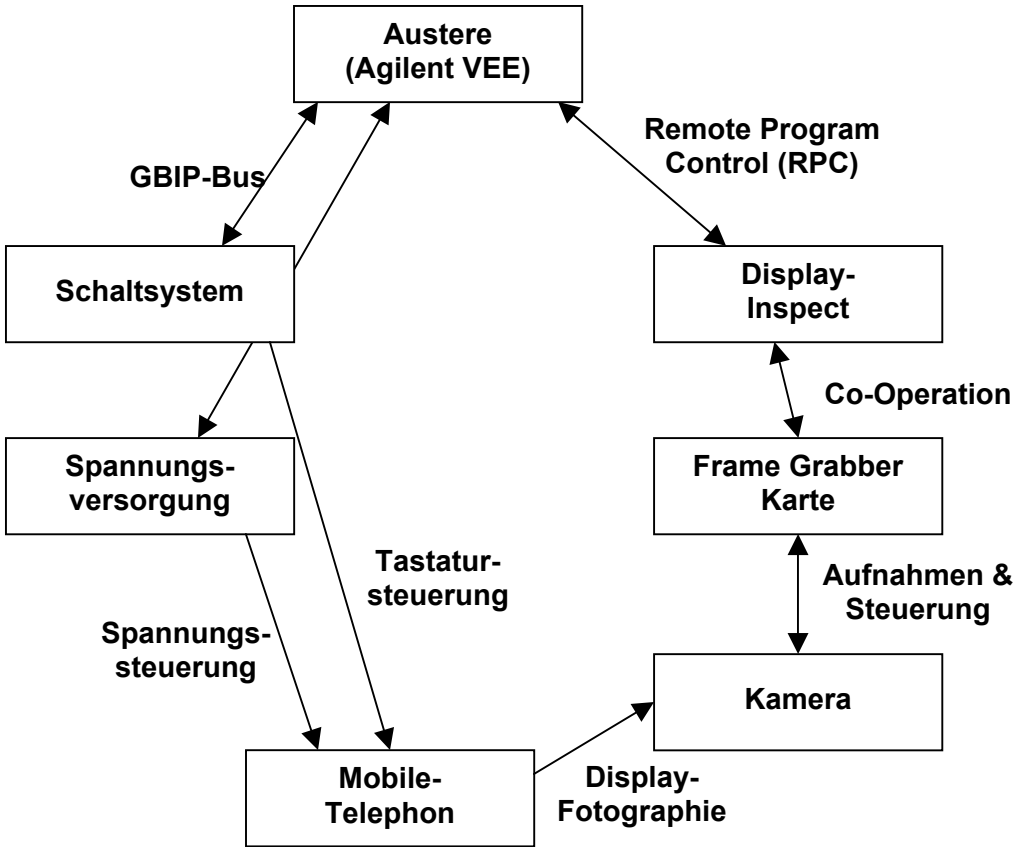


Abbildung 3.10: Zusammenhang der Komponenten untereinander

Der Robot Aided Phone Tester ist eine Bibliothek von Hochsprachenfunktionen, welche das zu testende Gerät über das Machine Vision System „DisplayInspect“, das Messdatenerfassungs-/ Schaltsystem (Switching Unit) und die Spannungsversorgung kontrolliert. Dieses System wurde unter der Programmierumgebung von VEE realisiert. VEE zeichnet sich unter anderem dadurch aus, dass es wegen seiner graphischen Programmierung einfach und leicht handhabbar ist, GPIB-Geräte, ActiveX Komponenten und dynamische Bibliotheken (shared objects unter Unix oder DLLs unter Windows) leicht anbinden und ansprechen kann. Fast alle Windows-Funktionen können sozusagen ausgenutzt werden. Des weiteren besitzt VEE eine abgespeckt integrierte Matlab-Laufzeitumgebung inklusive der DSP-Toolbox. Durch diese Eigenschaften können die Testskripte erheblich erweitert werden.

Das Schaltsystem und die Spannungsversorgung besitzen eine GPIB-Schnittstelle und sind über den GPIB-Bus an eine PCI-GBIP-Karte des Computer angeschlossen und erhalten hierüber Befehle und Anfragen. Die Relais des Schaltsystems sind wiederum mittels Klingeldrähten mit der Tastatur des Mobiltelephons verbunden, um Tastendrücke zu simulieren. Die Spannungsversorgung ist ebenfalls über den GPIB-Bus steuerbar. Sie kann die Spannung oder den Strom des Mobiltelephons beeinflussen oder bei fester Spannung den aufgenommenen Strom messen.

Auf der linken Seite stehen die beiden Komponenten zur Steuerung des Gerätes mit Eingaben: die Spannungsversorgung und das Schaltsystem. Auf der anderen Seite stehen

die Kamera, gesteuert über die Frame-Grabber-Karte und das Programm DisplayInspect. Diese Komponenten überwachen die Ausgaben des Bildschirms des Telephons.

Die Kamera bekommt ihre Steuerbefehle über ein Bussystem auf BNC-Basis und gibt ihre Daten über ein RGB-Kabel zurück an die Frame-Grabber-Karte. Diese Karte wird wiederum angesteuert durch DisplayInspect. Dieses Programm benutzt die Karte für Bildaufnahmen und kann diese über entsprechende Werkzeuge verifizieren. Im nächsten Kapitel werden diese kurz angesprochen. DisplayInspect wird wiederum ferngesteuert über die sogenannten Remote Program Controls (RPCs). Diese sind eine Sammlung von Bibliotheksfunktionen, über welche andere Programme (hier VEE) DisplayInspect „fernsteuern“ können. Innerhalb des Austere-Systems werden diese Eigenschaften ausgenutzt zur Aufnahme eines Bildes, Speicherung eines Bildes und einer Verifizierung zwischen zwei Bildern.

3.5.3 RAPT

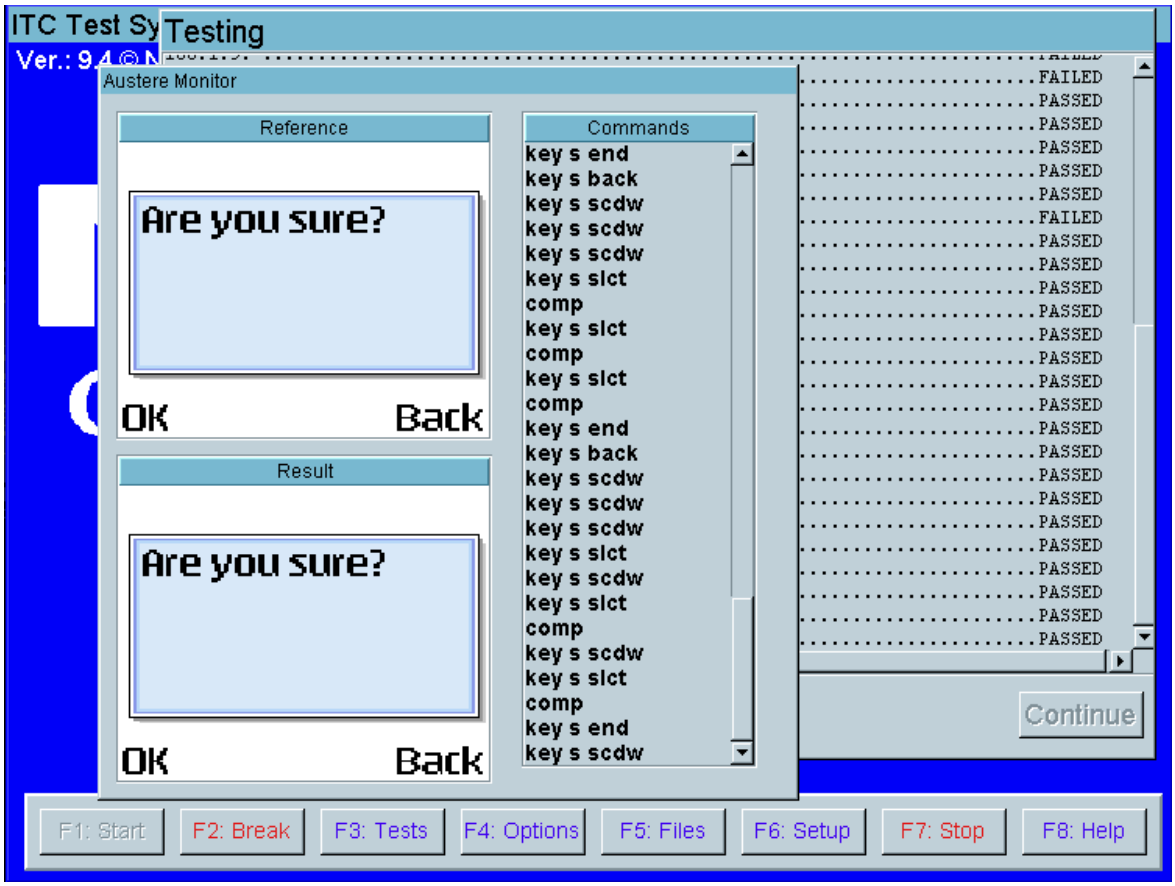


Abbildung 3.11: Ansicht von Austere zum Zeitpunkt der Ausführung von Testfällen

Aus objektorientierter Sicht besteht das Objekt RAPT aus den Methoden zur Ansteuerung der elektrischen Geräte über den GPIB Bus, den Methoden zur Fernsteuerung des Vision Systems DisplayInspect und den Methoden für den Datenbankzugriff. Die Eigenschaften des Systems sind in der Datenbank und einer Einstellungsdatei abgelegt. In der Datenbank stehen die Testfälle, und in der Einstellungsdatei stehen die grundlegenden Einstellungen für

das unter Test stehende Telephon, sowie spezifische Einstellungen für den Ablauf von Austere.

Eine typische Testfallausführung sieht wie folgt aus:

- Überführung der Referenzobjekte aus der Datenbank in das Testsystem,
- Ausführung der Kommandos des Testfalls (in meisten Fällen nur Drücken der Tasten),
- Vergleich des Resultates (Bild oder Ton) mit der Referenz und
- Zurückschreiben des Ergebnisses in die Datenbank (plus Urteil: passed/failed/manual)

Im Abbildung 3.11 ist beispielhaft eine Ausführung von Testskripten mit Austere zu sehen. Das Schaltsystem führt die Kommandos zur Simulation der Tastendrücke eines Testschrittes (Steps) aus, und mit DisplayInspect können die Bilder unter „Reference“ und „Result“ abschließend ausgewertet werden. Im Austere-Monitor-Fenster werden die beiden Bilder dann dargestellt. Im fortlaufenden Testing-Fenster (im Hintergrund) erscheinen neben den Test-IDs und Kurzbeschreibungen (linksseitig) die Auswertungsergebnisse oder Resultate der abgeschlossenen Testschritte (rechtsseitig: „passed“, „failed“ oder „manual“).

3.6 DisplayInspect

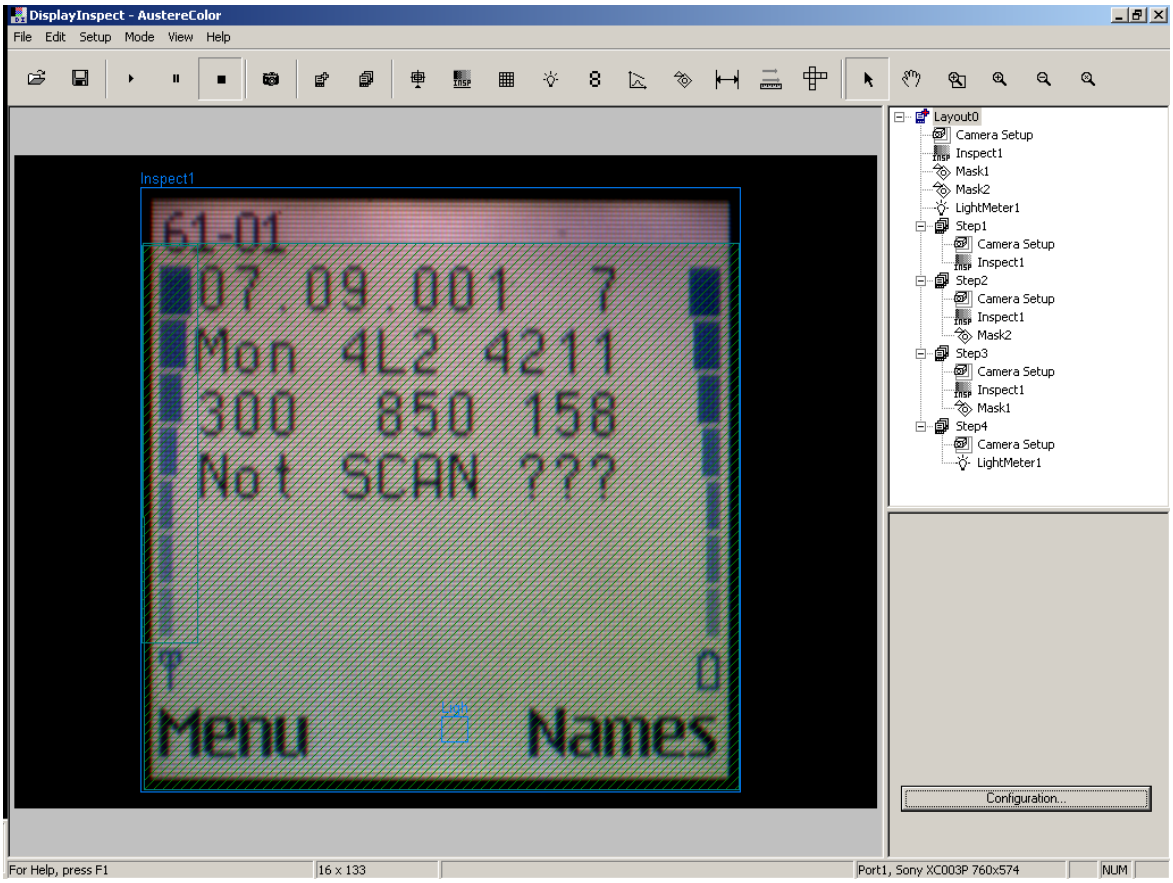


Abbildung 3.12: Cognex’ DisplayInspect mit geladenem Projekt

DisplayInspect ermöglicht die Aufnahme, die Präsentation und die Analyse von Bildern eines Displays, welches innerhalb von Tests aufgenommen wurde.

Jedes Bild passiert eine bestimmte Menge vorkonfigurierter Werkzeuge, welche folgende Aufgaben erfüllen:

- Vergleich von gefundenen und erwarteten Charakteristiken innerhalb eines Bildes,
- Verifizierung, ob die Charakteristiken in der richtigen Farbe erscheinen,
- Überprüfung der Ausrichtung von Charakteristiken gegenüber dem Bildrand und
- Sicherung der korrekten Funktion einzelner Pixel eines Bildes.

Die Ergebnisse, welche aus den entsprechenden Werkzeugen resultieren, können dazu benutzt werden, um festzustellen, ob Tests am Ende korrekt oder inkorrekt verliefen. Anschließend werden die Resultatdaten, welche auf den eingestellten Parametern basieren, zurückgeliefert.

Jedes DisplayInspect Werkzeug führt unterschiedliche Funktionen auf der eingestellten Zielfläche aus. Wird ein Werkzeug, wie z.B. eine Inspect Region, ein Pixel Probe, ein Light Meter oder eine Mask Region definiert, so müssen für die Analyse eine entsprechende Region und die erforderlichen Parameter dazu definiert werden. Folgend noch ein Kurzüberblick der wichtigsten Funktionen:

Werkzeug	**Funktion**
Inspect Region	Eine Inspect Region vergleicht die Qualität einer Region eines unter Laufzeitbedingungen aufgenommenen Bildes mit der entsprechenden Region eines Referenzbildes. Hier können defekte Charakteristiken im Display erkannt werden.
Pixel Probe	Ein Pixel Probe inspiziert individuelle Pixel des Aufnahmebildes. Es kann dazu benutzt werden, um festzustellen, ob jedes Pixel in einer Region ordentlich funktioniert.
Light Meter	Ein Light Meter misst entweder die Helligkeit eines Schwarzweiß-bildes oder den Farbwert einer Region des Bildes. Dieses Werkzeug prüft, ob die Hintergrundbeleuchtung funktioniert oder jede Einzelheit in der richtigen Farbe dargestellt wird.
Alignment Mark	Ein Alignment Mark findet die genaue Position eines bekannten Referenzpunktes des Bildes. An diesem Punkt können die anderen Werkzeuge durch Verschiebung ausgerichtet werden.
Mask Region	Eine Mask Region gibt eine bestimmte Region an, die durch andere Werkzeuge nicht analysiert werden sollen.

All diese Werkzeuge innerhalb eines DisplayInspect Projektes werden in Layouts gruppiert. Hier können alle relevanten Werkzeuge zur Analyse eines Bildes eingefügt werden. Ein Layout wird meist in Zusammenhang mit einer Kamera benutzt. Sind mehrere Kameras eingebunden, so lohnt sich die Einführung weiterer Layouts. Steps enthalten wiederum eine ausgewählte Sammlung von Werkzeugen aus einem Layout. Ein Step wird meist verwendet, um während einer Analysephase eines Bildes separate Charakteristiken zu testen. Zum Beispiel wird ein Step dazu benutzt, ein ganzes Bild zu verifizieren, ein anderer Step, um nur einen Teil des Bildes und die Hindergrundbeleuchtung zu testen.

4 AUSTERE-ERWEITERUNGEN

In diesem Kapitel sind alle Erweiterung aufgeführt, welche während der Diplomarbeit erarbeitet worden sind. Die Ausnahmen bildet der Vergleich von Tönen oder Melodien und das Kapitel über Schrift- und Zeichenerkennung. Diese Erweiterungen wurden wegen der Ausführlichkeit in die Kapitel 5 beziehungsweise 6 ausgegliedert. Ein Kurzüberblick über schon vorhandene Funktionen befindet sich im Kapitel 4.1.1 und über die Erweiterungen im Kapitel 4.1.2.

4.1 Kommandoüberblick

Die Skripte des Austere-Systems untergliedern sich in Kommandos. Diese Kommandos erfüllen wie in jeder anderen Programmiersprache auch eine bestimmte Funktion. Im Unterpunkt 4.1.1 sind die schon implementierten Befehle zu sehen. Und im Unterpunkt 4.2.1 werden die neu hinzugekommenen Befehle kurz erläutert.

4.1.1 die am häufigst gebrauchten Befehle von Austere

- **key *type key [no]***: löst über das Schaltsystem einen Tastendruck der Länge *type* auf die Taste *key* des Telephons *no*-mal aus.
- **comp**: lädt das Referenzbild in DisplayInspect und trainiert es; anschließend wird eine Bildaufnahme gemacht, welche automatisch mit dem trainierten Bild verglichen wird; das Resultat („passed“ oder „failed“) und das gegebenenfalls fehlerhafte Resultatbild werden abschließend gesichert.
- **shoot**: nimmt einen Schnappschuss vom Display auf und gibt das Resultat „manual“ zurück, solche Aufnahmen müssen von Hand verifiziert werden.
- **macro *macroID***: löst die Kommandos innerhalb des Makros *macroID* auf; Makros können bis zu einer Tiefe von Vier ineinander verschachtelt werden.
- **mkcall**: veranlasst den GSM-Netzwerksimulator, einen Anruf zum Telephon zu simulieren.
- **discall**: veranlasst den Simulator, die vorhandene Verbindung wieder abzubauen.
- etc.

4.1.2 die Erweiterungen des Befehlssatzes in Kurzübersicht

- **tckwait**: hält das Austere-System während der Skriptabarbeitung solange an, bis der nächste TCK-Test ordnungsgemäß aufgeladen worden ist. Auf diesen Befehl wird im Kapitel 4.4 näher eingegangen.

- **yesno**: bestätigt die Endabfrage eines TCK-Tests. Prüft zunächst die Testergebnisse der letzten Vergleiche und nimmt die abschließende Validierung vor. Nähere Erläuterungen finden sich im Kapitel 4.5.

- **set *variable value***: wurde zunächst nur zum Setzen der Länge eines Tastendruckes verwendet. Mit einer Änderung können nun nahezu alle Variablen während der Abarbeitung der Skripte verändert werden. Mehr dazu ist im Kapitel 4.6 aufgeführt.

- **shoot [*time*]**: besitzt zwei Abarbeitungspfade. Ohne Angabe des Parameters *„time“* wird ein Schnappschuss vom Display aufgenommen. Mit der Parameterangabe nimmt shoot eine Reihe von Schnappschüssen im vorher angegebenen Intervall auf und generiert eine Videosequenz. Auf diesen Befehl wird im Kapitel 4.7 näher eingegangen.

- **sndcmp *time***: nimmt einen Ton (meist eine bestimmte Frequenz mit bestimmter Dauer) oder ein Klang (meist ein kleines Musikstück, die in der Galerie des Telephons zu finden sind) auf und vergleicht ihn mit der Referenz. Diesem Thema wurde das Kapitel 6 gewidmet.

- Die unabhängige Schrift- und Zeichenerkennung wird noch nicht im Austere-System mittels eines Kommandos verwendet.

4.2 Überblick über MIDP, TCK & co.

Ein Ziel dieser Diplomarbeit ist es, das bestehende Austere-System um die Möglichkeiten zu erweitern, die integrierte Java Laufzeitumgebung (MIDP) abtesten zu können. Dies geschieht bis dato manuell und soll Schritt für Schritt automatisiert werden. Um dies zu verwirklichen setzen an dieser Stelle die Erweiterungen um die Befehle oder Kommandos „tckwait“ und „yesno“ an (Kaptitel 4.3 und 4.4).

Das Mobile Information Device Profile (MIDP) ist ein Menge von APIs (ein Profil) für die Java 2 Platform, Micro Edition (J2ME). MIDP ist für kleine Geräte mit eingeschränkten Ressourcen wie Mobiltelephone und PDAs konzipiert. Diese APIs und weitere spezifische, eigenentwickelte APIs von Nokia werden seit längerem in den meisten Mobiltelephonen von Nokia integriert.

Auf dem Mobiltelephon muss für die Tests ein sogenanntes Midlet vorhanden sein. Wird das Midlet gestartet, übernimmt es die Kommunikation mit einem verbundenen PC, indem es auf die Übertragung eines Javatests in Form eines TCK-Testprogramms in ausführbarer, binärer Form wartet. Wurde eines übertragen, so wird es zur Ausführung gebracht und nach Beendigung übernimmt das Midlet die Rückübertragung des Resultats. (TCK bedeutet Technology Compatibility Kit) In jedem TCK-Testprogramm wird auf eine spezielle Fähigkeit von MIDP eingegangen, z.B. auf das Testen der ordentlich formatierten Ausgabe langer Zeichenketten oder die Darstellung und Handhabung von Listboxen oder Auswahlgruppen.

Auf der PC Seite muss das Programm JavaTest gestartet werden, welches einen Kommunikationsserver bereitstellt ähnlich eines Webservers. Eine Brücke übernimmt schließlich die Kommunikation zwischen diesem Server und meist einer seriellen Schnittstelle mit Anschluss an ein Mobiltelephon. Das JavaTest Programm bietet unter anderem die Möglichkeiten:

- einzelne Tests aus einer ganzen Reihe oder alle auszusuchen,
- den Ausführungsstatus der Resultate und Übertragungen anzuzeigen und
- die Resultate aus den beendeten Tests in Form von HTML-Seiten für spätere Auswertungen aufzubereiten.

JavaTest wird von Sun Microsystems für die Tests von MIDP bereitgestellt und kann von deren Homepage aus kostenlos geordert werden. Hier lassen sich auch weitere Einzelheiten in Erfahrung bringen. Abbildung 4.1 zeigt JavaTest vor Abarbeitung von TCK-Tests.

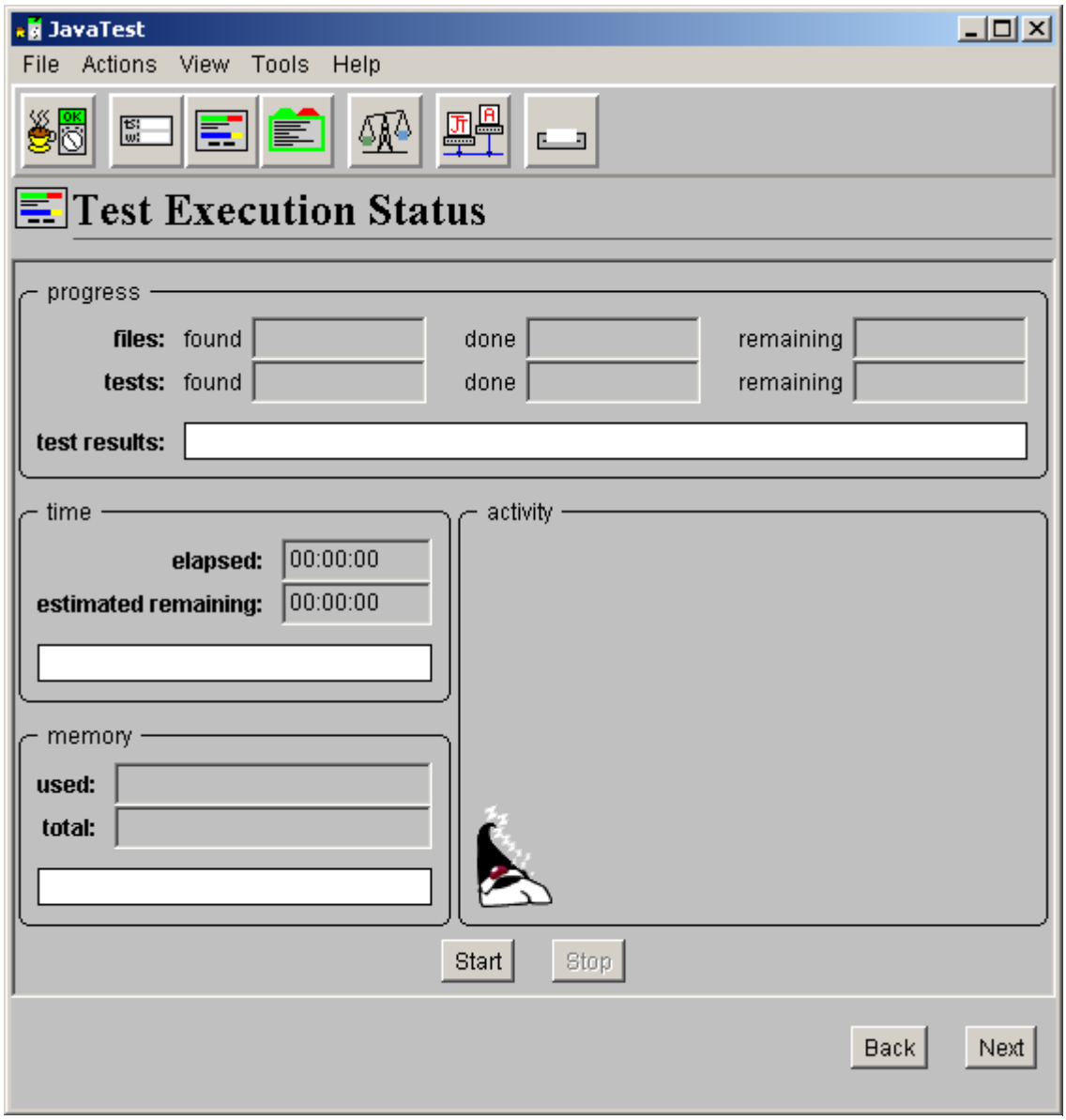


Abbildung 4.1: JavaTest vor Ausführungsbeginn

4.3 TCKWAIT

Mit dem Kommando „tckwait“ soll es ermöglicht werden, die laufende Kommandoausführung der Austere-Testskripte solange anzuhalten, bis das nächste TCK-Testprogramm aufgeladen worden ist. In Abbildung 4.2 ist die Ausprogrammierung des Kommandos „tckwait“ zu sehen.

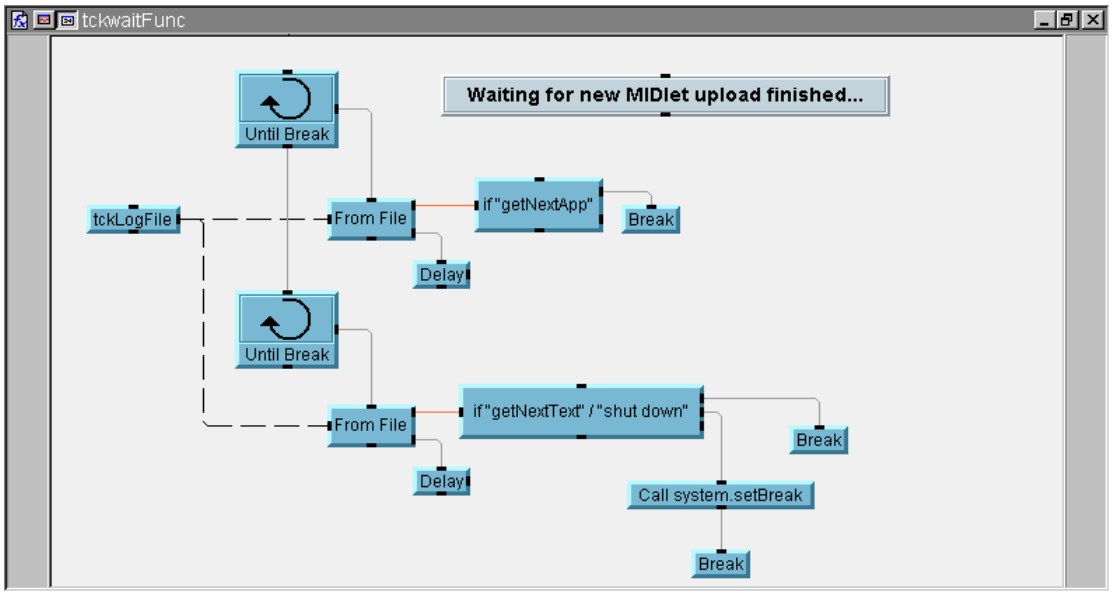


Abbildung 4.2: Ausprogrammierung der Funktion *TCKWAIT*

Das Programm JavaTest ist selbst ein in Java entwickeltes Programm. Es wird innerhalb einer Kommandokonsole von Windows gestartet und gibt auf dieser eine fortschreitende Protokollierung über die Kommunikation mit dem Midlet vom Mobiltelephon aus. Die Tckwait-Funktion arbeitet auf Basis der Abfrage dieses Protokolls. Nachfolgend die Erläuterung des Protokolls und die der tckwait-Funktion.

Zunächst muss vor jeder Testreihe der Kommunikationsserver gestartet werden:

```
com.sun.cldc.communication.midp.MIDHttpExecutionServer started @ port 1907
com.sun.cldc.communication.midp.MIDHttpSupportServer started @ port 8089
```

Wird anschließend das Midlet auf dem Telephon gestartet, stellt es sofort eine Anfrage nach dem Namen des nächsten TCK-Testprogrammes:

```
got new request: GET /test/getNextApp.jad HTTP/1.1
getNextApp
```

Der Name „test1.jar“ wird zurück zum Midlet geschickt:

```
next_app: test1.jar
jar_path: c:\apps\tck\MIDP-TCK_10a\temp\test1.jar
```

Anschließend fordert das Midlet das neue Programm an:

```
got new request: GET /test1.jar HTTP/1.1
file download: c:\apps\tck\MIDP-TCK_10a\temp\test1.jar
```

Mit der Anfrage nach einem neuen Test ist das Herunterladen des Programms beendet:

```
got new request: GET /test/getNextTest HTTP/1.1
getNextTest
```

Nach dem Durchlauf des Testprogramms verlangt das Midlet, das Resultat des beendeten Tests senden zu dürfen:

```
got new request: POST /test/sendTestResult HTTP/1.1
sendTestResult
got new request: GET /test/getNextTest HTTP/1.1
getNextTest
```

Werden mehrere Tests unter JavaTest ausgesucht, so folgen die Anfragen des Midlets immer nach gleichem Muster:

```
got new request: GET /test/getNextApp.jad HTTP/1.1
getNextApp
next_app: test2.jar
jar_path: c:\apps\tck\MIDP-TCK_10a\temp\test2.jar
got new request: GET /test2.jar HTTP/1.1
file download: c:\apps\tck\MIDP-TCK_10a\temp\test2.jar
got new request: GET /test/getNextTest HTTP/1.1
getNextTest
got new request: POST /test/sendTestResult HTTP/1.1
sendTestResult
got new request: GET /test/getNextTest HTTP/1.1
getNextTest
```

Nachdem alle ausgesuchten TCK-Testprogramme aufgeladen wurden, wird der Server wieder heruntergefahren. Das Midlet beendet sich im Gegensatz erst nach manuellem Schließen.

```
com.sun.cldc.communication.midp.MIDHttpExecutionServer shut down.
com.sun.cldc.communication.midp.MIDHttpSupportServer shut down.
```

Dieses Protokoll wird während des Starts von JavaTest in eine Datei umgeleitet und kann von Austere Zeile für Zeile kontrolliert ausgelesen und analysiert werden. Das Aufladen einer Applikation zum Mobiltelephon wird initiiert durch die Antwort der Anfrage „getNextApp“ und der Datentransfer beginnt. Das Aufladen ist beendet mit der Antwort auf die Anfrage von „getNextTest“.

Hieraus bot sich eine kaskadische Programmierung an. Während des ersten Schleifendurchlaufs werden ab aktuellem Dateizeiger alle nachfolgenden Zeilen auf Vorkommen von „getNextApp“ geprüft. Während in der zweiten Kaskade auf das Ende des Ladevorganges gewartet wird. Bei Erscheinen der Zeile mit dem Inhalt „getNextTest“ beendet sich diese Funktion. Sie stellt daher sicher, dass erst nach ordentlichem Download mit dem Testen fortgefahren wird. Ein kompletter Abbruch des Testdurchlaufs sollte nur dann geschehen, wenn ein Server frühzeitig heruntergefahren wurde und somit das Testen beendet worden ist.

Ohne Hilfe über diese Protokollierung müsste eine maximale Wartezeit eingeführt werden. Jedes Laden einer TCK-Applikation variiert je nach Größe, Systemauslastung und unterschiedlicher Prozesszeitvergabezeiten durch das Betriebssystem. Eine konstante Wartezeit schließt sich somit aus.

4.4 YESNO

Nachdem ein TCK-Testprogramm durchlaufen und mit allen Einstellungsmöglichkeiten und ggf. Variationen abgearbeitet wurde, erscheint auf dem Display eine abschließende Frage nach der Richtigkeit oder Korrektheit aller im laufenden Test abgetesteten Funktionalitäten. In einer abschließender Frage könnte beispielsweise gefragt werden, ob der Schriftsatz in allen Variationen (groß, klein, unterstrichen, kursiv, etc.) korrekt angezeigt worden ist, oder ob ein überlanger Text über Navigationshilfen auf dem Display korrekt dargestellt wurde. Hier besteht die Wahl der Antwort auf die abschließende Frage zwischen einem Ja oder einem Nein. Die Eingabe der englischen Antwort „Yes“ oder „No“ geschieht mittels Tastendruck auf die linke respektive rechte Softkeytaste.

Für die Umsetzung auf automatisch ablaufende Tests bedeutet dies, dass alle vorherigen Ergebnisse von Vergleichen innerhalb des Testskripts mittels der Vergleichsbefehle „comp“ oder „sndcmp“ für die Entscheidung herangezogen werden müssen. Im Gegensatz dazu resultiert der Befehl „shoot“ immer in dem Ergebnis „manual“, da dieser nur Schnappschüsse vom Display nimmt und keinen Vergleich nach sich zieht. Aus diesem Grund sollte dieser Befehl bei TCK-Tests nicht verwendet werden.

Nach Analyse der Resultate von „comp“ fand sich eine einfache Lösung: die Einführung eines einfachen Zählers. Für positive Vergleiche, in welchen Resultat und Referenz übereinstimmen, werden in der Datenbank das Ganzzahlattribut „0“ und das zugehörige Textresultat „passed“ abgelegt. Für ein negatives Ergebnis eines Vergleichs werden die Attribute „1“ und „failed“ abgelegt. Das Ergebnis des Kommandos „shoot“, liefert immer einen negativen Wert, und sollte deshalb vermieden werden. Der hier eingeführte Zähler „yesnocnt“ setzt vor Ablage der Ergebnisse in die Datenbank an. Er wird mit dem Ergebnis des Vergleichs hochgezählt. In Abbildung 4.3 ist dies verdeutlicht. Die Ausprogrammierung der Funktion „yesnoFunc“ ist in Abbildung 4.4 zu sehen. Nach Aufruf der Funktion wird der Zähler „yesnocnt“ abgefragt und in dessen Abhängigkeit wird die entsprechende Softkeytaste durch das Schaltsystem betätigt. Lag während eines TCK-Tests mindestens ein Fehler vor, wird mit dem Schalter 313 des Schaltsystems der rechte Softkey („No“), andernfalls mit 314 der linke Softkey („Yes“) des Telephons betätigt. Abschließend wird dieser Zähler wieder rückgesetzt.

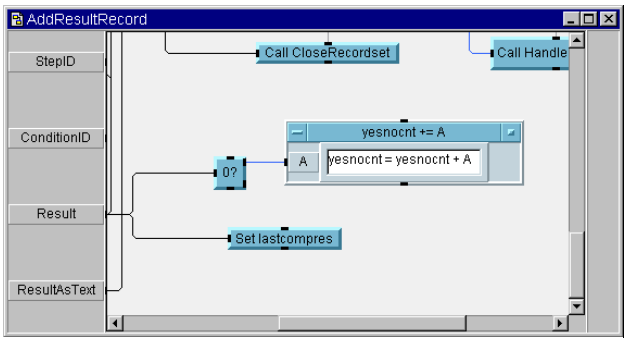


Abbildung 4.3: Resultatspeicher „yesnocnt“ für Endabfragen von TCK-Tests

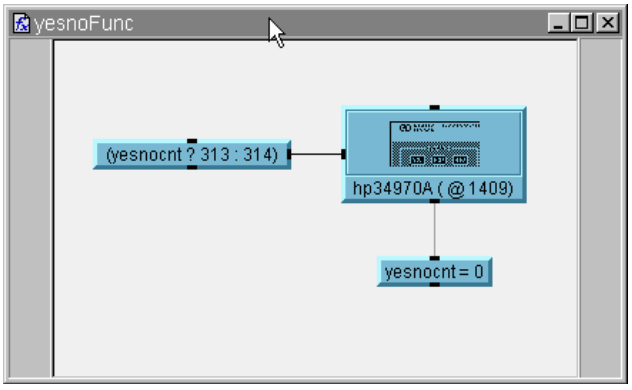


Abbildung 4.4: Softkey Tastendruck in Abhängigkeit des Zustands von „yesnocnt“

Hier stellt sich eine neue Frage, nämlich die, wie Testskripte von TCK-Tests und Nicht-TCK-Tests kombiniert werden können, ohne dass der erste TCK-Test negativ ausfällt. Das „set“-Kommando, welches im nächsten Kapitel erläutert wird, kann dazu benutzt werden.

4.5 SET *variable value*

Die Änderung des „set“-Kommandos schien die logische Konsequenz des Problems der Kombination zwischen normalen Tests und den speziellen TCK-Tests. Dieses Kommando wurde wegen der in Kapitel 4.4 genannten Anforderung erweitert.

Alle im Austere-System befindlichen Variablen können mit dieser Struktur der Ausprogrammierung nun direkt manipuliert und verändert werden. Dies bedeutet für den Testingenieur beim Schreiben von Skripts einerseits einen ernormen Vorteil der Konfigurierbarkeit des Systems während der Testausführung, aber andererseits auch die Gefahr bei Veränderungen von Variablen innerhalb von Testschritten unwissentlich zum Schaden des Systems beizutragen. Hier ist Achtung geboten.

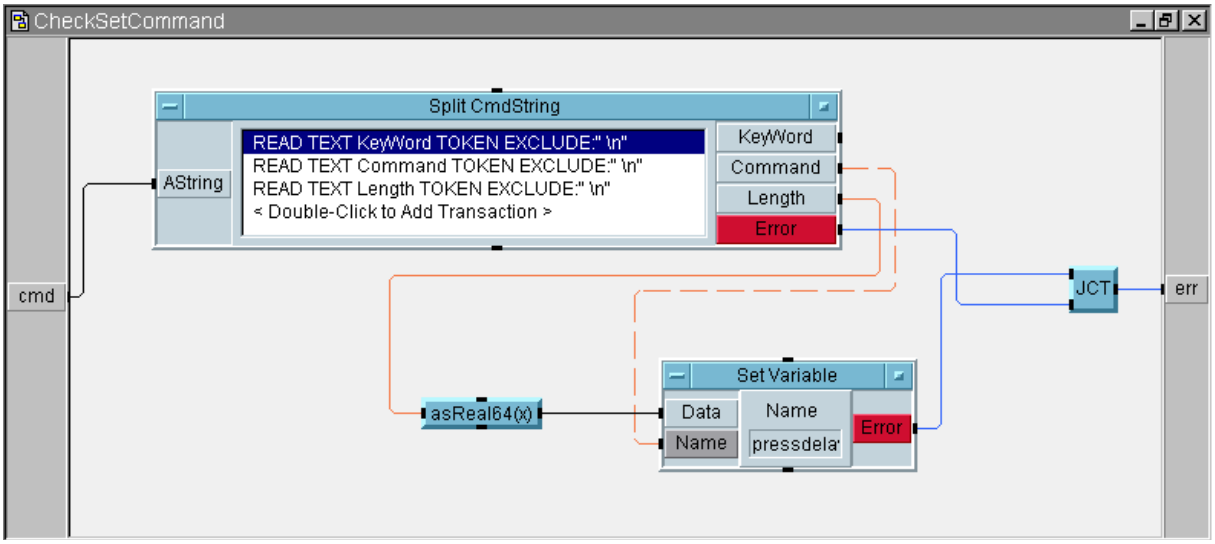


Abbildung 4.5: das „set“-Kommando setzt eine gegebene Variable auf einen bestimmten Wert

Vor der Änderung des Kommandos wurden lediglich die Längen der Tastendrücke und die Abstände zwischen diesen verändert. Das Kommando war starr und nur darauf ausgerichtet. Unter dieser Ausführung ist es flexibler und offener zugleich.

In „Split CmdString“ wird das komplette Kommando getrennt in das Kommando selbst („KeyWord“), die zu setzende Variable (“Command”) und den zugehörigen Wert („Length“). Vor dem Setzen des Wertes wird dieser wegen interner Vereinheitlichung in einen Gleitkommawert umgewandelt („asReal64“).

4.6 SHOOT [*time*]

Das ursprüngliche Kommando hatte die Aufgabe, einen Schnappschuss zu machen und diesen ohne Vergleich als Resultat zu speichern. Das verbesserte Kommando shoot besteht aus zwei Funktionen: zum einen aus der Funktion zur Initialisierung und Generierung des Videos; zum anderen aus der Funktion, welche die Schnappschüsse macht, der zeitgetriggerten Funktion. Die neu hinzugekommene Funktionalität ergänzt die bestehende optional mit einem Argument, sodass kein Zwang der Änderung zu bestehenden Testskripten herrscht.

4.6.1 Die zeitgetriggerte Funktion von shoot

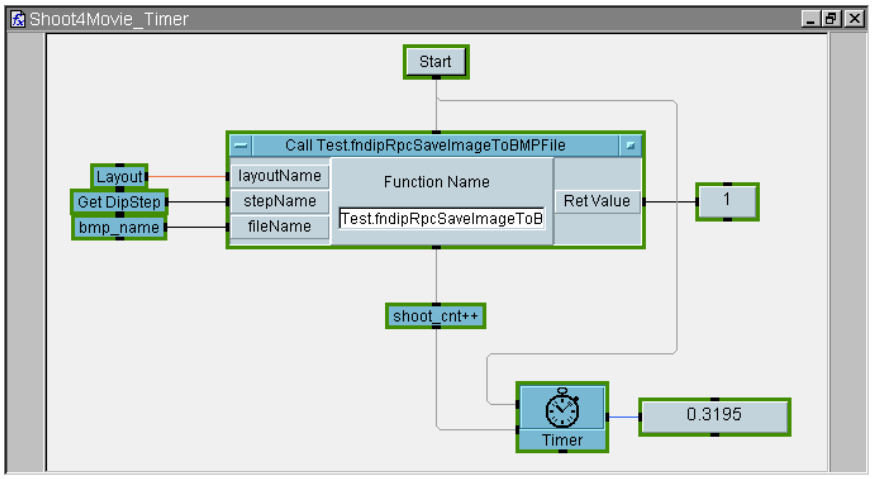


Abbildung 4.6: die zeitgetriggerte Funktion von shoot

Diese Funktion nutzt die Eigenschaften von DisplayInspect dahingehend aus, dass DisplayInspect veranlasst wird, ein Bild von der Kamera aufzunehmen und anschließend unter angegebenem Dateinamen („bmp_name“) abzuspeichern. Dies geschieht programmiertechnische mit der „SaveImageToBMPFile“-Funktion. Anschließend wird der Bildzähler hochgezählt.

4.6.2 Das ausführende „shoot“

Die Abfrage nach dem Vorhandensein des *time* Parameters geschieht unmittelbar vor Einstieg in die Funktion Shoot. Ist kein Parameter vorhanden, so wird der Funktion Shoot der Wert „0“ als Zeit übergeben. (Ohne ein einziges Bild ist es unmöglich, ein Video zu erstellen.) Die Entscheidung über den weiteren Ablauf der Funktion geschieht im ersten If-Else-Block: „if (count==0)“. Besitzt der Eingangsparameter *tm* den Wert „0“, so wird die Abarbeitung im Block GetResult fortgeführt. Hier wird vom Display ein Schnappschuss gemacht und dieser im gegebenen Dateinamen abgelegt.

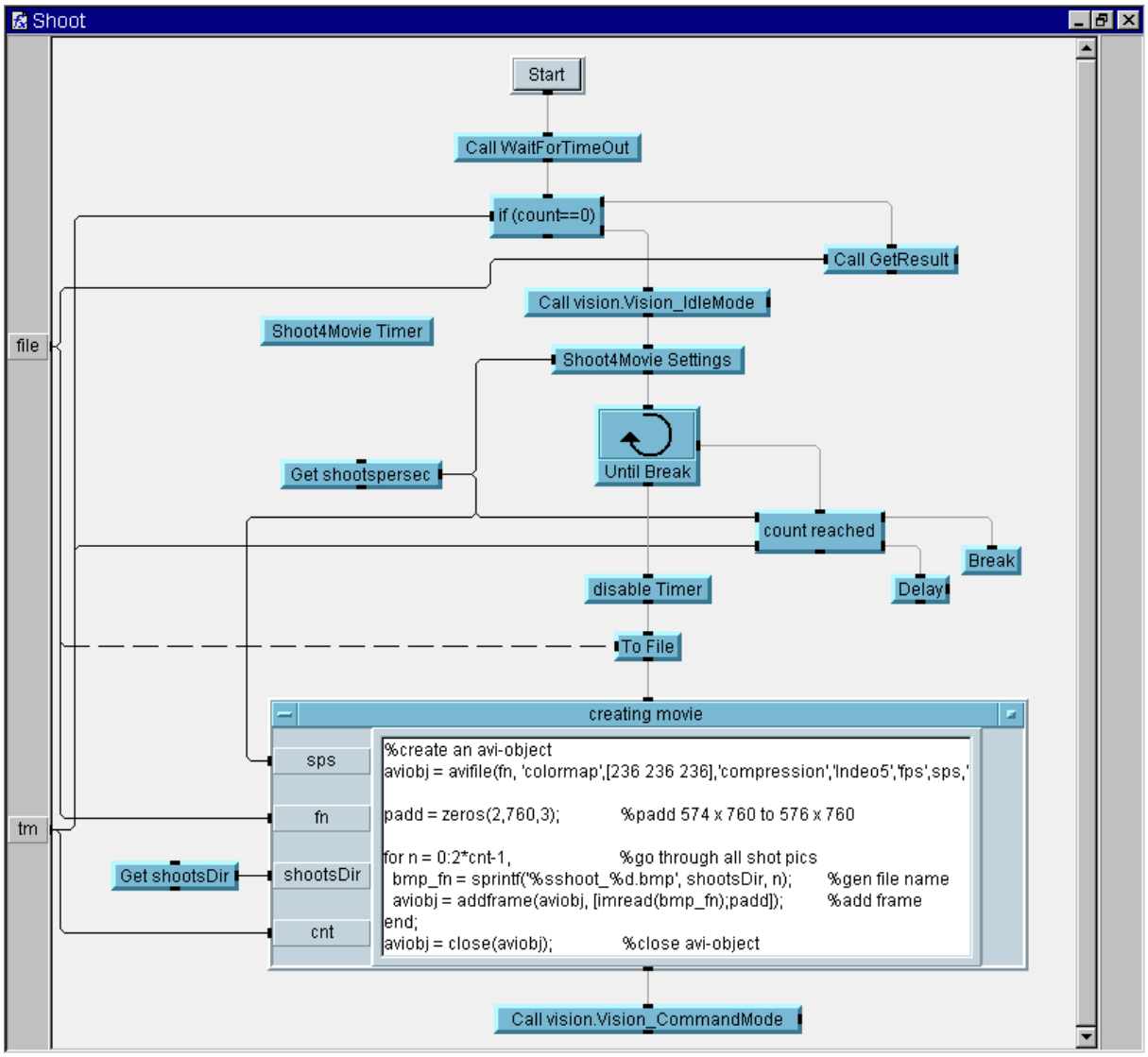


Abbildung 4.7: die Funktion des Kommandos „shoot [*time*]“

Im Falle der Angabe eines *time* Parameters, wird ein Zeitgeber (Shoot4Movie_Timer) so initialisiert, dass er, wenn nicht anders über die Variable „shootspersec“ angegeben, alle 500ms einen Schnappschuss vom Display macht und in einer Datei mit fortlaufendem Dateinamen abspeichert. Der Dateiname setzt sich aus dem Verzeichnisnamen „shootsDir“, dem Namen „fn“ und dem fortlaufendem Zähler zusammen. In der zeitgetriggerten shoot-Funktion läuft ein Zähler mit, mit welchem nachvollzogen werden kann wie weit die Ausführung fortgeschritten ist. Ist der Zähler abgelaufen – innerhalb der Schleife unter „Until Break“ mittels „count reached“ - so wird die zeitgetriggerte Funktion mit „disable Timer“ beendet. Ein gegebenenfalls schon vorhandene Videodatei wird intern des Blocks „To File“ gelöscht. Mit Hilfe von Matlab wird schließlich ein Video generiert. Das verwendete Dateiformat ist das AVI-Format (von Matlab fest vorgegeben). Der Komprimierungscodex lässt sich jedoch konfigurieren. Hier hat sich Indeo5 als ausreichend und allgemein verwendbar herausgestellt. Dieser reduziert die Einzelbilder so, dass der Speicherbedarf gering gehalten werden kann.

4.6.3 Das Matlab Script „creating movie”

Das Matlab Skript erstellt aus den Schnappschüssen ein Video. Die Eingangsparameter zu Matlab sind: die Anzahl der Bilder je Sekunde (sps), der Dateiname, in dem das Video abgelegt werden soll (fn), der Verzeichnisname, in dem die Schnappschüsse gespeichert wurden (shootsDir) und die Länge der Schnappschussaufnahmen (cnt) in Sekunden.

Das Video muss im vorhinein gelöscht werden, da die zusätzlichen Bilder im Verlauf durch „addframe“ angefügt werden. Ein neues oder bestehendes Videoobjekt wird zunächst geöffnet und konfiguriert. Dabei werden der Dateiname, die Farbtiefe, die Kompression, die Anzahl der Bilder und Schlüsselbilder pro Sekunde sowie die Qualität der Bilder angegeben. Bei der Angabe der Indeo-Kompression ist nur eine maximale Farbtiefe von 236 pro Farbkanal erlaubt. Über den Zeitgeber werden pro Sekunde nur wenige Vollbilder aufgenommen und dadurch, dass mindestens ein Vollbild Schlüsselbild sein muss, sind auch hier Grenzen bei der Kompression gesetzt.

```
aviobj = avifile(fn, ‘colormap’, [236 236 236], ‘compression’, ‘Indeo5’,
                     ‘fps’, 2, ‘keyframe’, 1);
```

Eine Restriktion des Indeo-Codex besteht darin, dass nur Bilder mit einer teilbaren Größe von 4 angefügt werden können. Die exportierten Bilder von DisplayInspect besitzen jedoch die Maße 760x574 und müssen erweitert werden. Die Variable `padd` wird so angelegt, dass die geforderte Teilbarkeit durch ein vorinitialisiertes Feld der fehlenden Größe erreicht wird.

```
padd = zeros(2, 768, 3);
```

In nachfolgender for-Schleife werden die Dateinamen der Bilder mittels `sprintf` generiert, mit `imread` geladen und mit `addframe` an das Video angehängt. Der Wertebereich des Schleifenzählers beginnt bei 0 und geht bis zur Anzahl der aufgenommenen Bilder minus 1.

```
for n=0:shootpersec*cnt-1,
    bmp_fn = sprintf(‘%sshoot_%d.bmp’, shootsDir, n);
    aviobj = addframe(aviobj, [imread(bmp_fn); padd]);
end;
```

Abschließend wird die Videodatei geschlossen und liegt zur Übertragung in die Datenbank bereit.

```
aviobj = close(aviobj);
```

Die Aufnahe eines kurzen Videos ist beendet. Diese Art der Aufnahme lohnt sich dann, wenn vom Telephon Meldungen von kurzer Dauer auf dem Display erscheinen sollen. Solche können besser durch ein kurzes Video erfasst werden, als durch einen einzigen Schnappschuss, welcher zum Unglück noch im falschen Moment geschossen werden könnte!

5 SCHRIFT-/ZEICHENERKENNUNG

5.1 Einleitung

Während der Entwicklung der Software werden von einem Build zum nächsten Veränderungen und Neuerungen vorgenommen. Diese werden ständig mittels Regressions- und Smoke-Tests überprüft. Nachteilig hierbei und beim Testen nach dem Black-Box-Verfahren allgemein, ist das Wissen um manche internen Werte der Software. Über den Menüpunkt „Net Monitor“ können die wichtigsten von ihnen angezeigt und kontrolliert werden. Diese Werte können helfen, mögliche Fehlerursachen schneller einzugrenzen und zu finden.

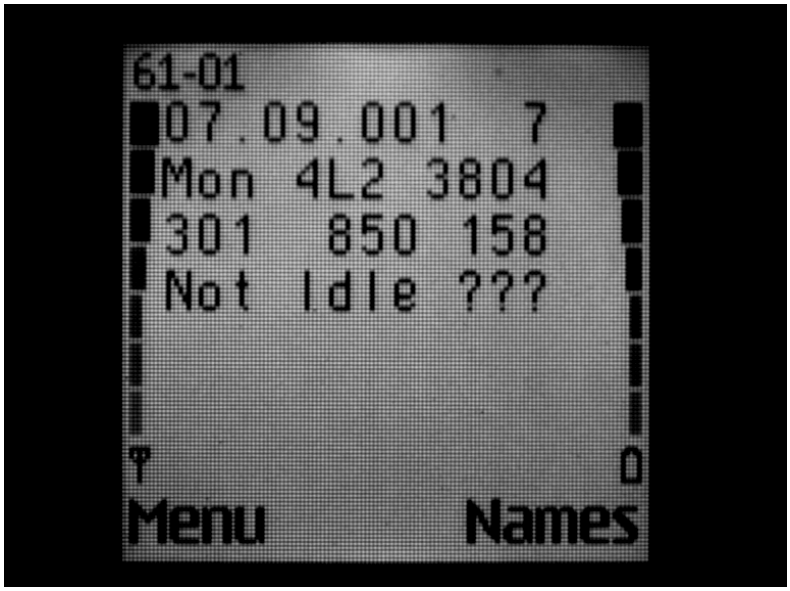


Abbildung 5.1: Screenshot des Net Monitors mit der Seite 6101.

Die Analyse aller Angaben des Net Monitor würde diese Arbeit jedoch sprengen, deshalb sei hier nur eine beispielhaft erläutert: auf der Seite 61-01 kann in der zweiten Zeile rechts die aktuelle Spannung der Batterie abgelesen werden: 3804 mV. Nach dieser Spannung richtet sich u.a. der Ausschlag des Balkens auf der rechten Seite des Displays. Der Ausschlag zu diesem Zeitpunkt besitzt eine Balkenhöhe von 7 (Angabe in der erste Zeile rechts).

Das Austere-System kann diese Werte jedoch nicht direkt aus dem Telephon lesen und verarbeiten, es hat nur die Möglichkeit des Vergleichs von ganzen Bildern oder Ausschnitten daraus. Der Gedanke der Einführung einer Zeichen- und Schriftenerkennung innerhalb eines eigenen Testprogramms liegt nahe. Nach dem Studium der existierenden Möglichkeiten ergab sich eine Lösung mittels eines einfachen Feed-Forward-Netzwerks.

5.2 Allgemeines zur Mustererkennung

Bei dem Wort Mustererkennung denkt man allgemeinhin an die korrekte Abbildung von Muster auf Klassen. Die Aufgabe einer computerbasierten Mustererkennung ist es, eine transparente Abbildung zu schaffen und diese einem Computer zu beschreiben.

In Abbildung 5:2 ist eine abstrakte Abbildung eines Objekts X zu sehen, welches mehrere Instanzen $\underline{x}$ haben kann. Diese nehmen meist unterschiedliche Gestalt an, gehören aber zur gleichen Klasse. Die Gesamtprozedur der Erkennung wird beschrieben durch die Abbildung $R(f(\underline{x})) \rightarrow c(\underline{x}) = X$. Diese setzt sich aus zwei Schritten zusammen. Im ersten Schritt werden die speziellen Muster des Objekts X durch seine Merkmale beschrieben, das ist der Weg von $\underline{x}$ nach $f(\underline{x})$. Im zweiten Schritt wird eine eindeutige transparente Abbildung $R(f(\underline{x}))$, also von $f(\underline{x})$ nach $c(\underline{x}) = X$ ausgeführt. Doch zunächst herrscht noch Unklarheit über die Funktion $f(\underline{x})$. Diese muss für die Erkennung vorberechnet werden.

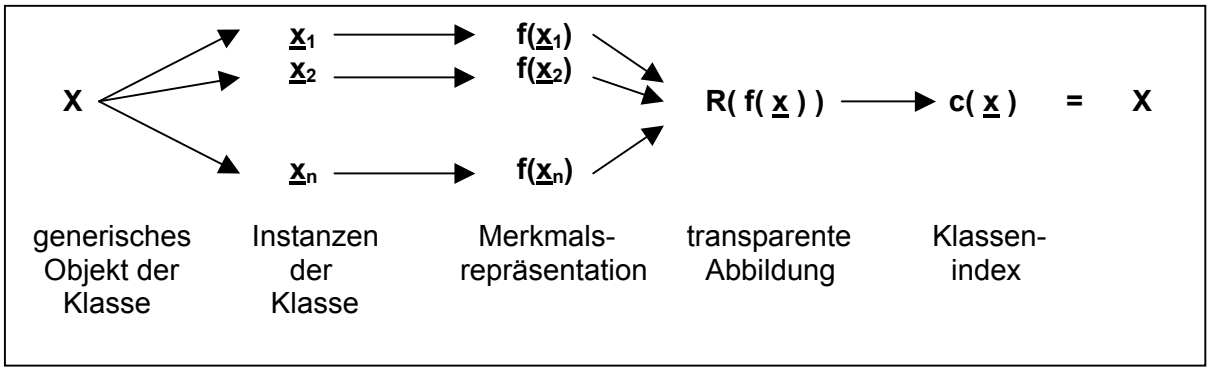


Abbildung 5.2: Klassenrepräsentation durch ein neuronales Netzwerk

In unserem Fall der Zeichen-/Schrifterkennung geht es darum, eine Ähnlichkeit eines Objekts mit einem vorgegebenem Muster herauszufinden, sozusagen eine Klassifikation eines Musters vorzunehmen. Das bedeutet, dass ein neues Muster der Klasse des Objekts *A* zugeordnet werden muss, falls es die größte Ähnlichkeit zu einem Muster besitzt, welches zur Klasse *A* gehört. Dies gelingt auf eindeutige Weise, falls sich die Muster der Klasse nicht mit einem Muster einer anderen Klasse überschneiden. Überschneiden sie sich jedoch in einigen Merkmalen, so muss vom besten Ergebnis ausgegangen werden, obwohl die Aussage über das zweitbeste Ergebnis vielleicht sogar nach menschlichem Ermessen richtiger wäre.

In nachfolgenden Abschnitten wird anhand dieses Models die Umsetzung erläutert. Im einzelnen:

- die Extrahierung der Merkmale,
- das Lernen der transparenten Abbildung R() und
- das Testen der eingeübten Klassifikation.

Das zentrale Konzept der Mustererkennung beruht auf der Diskriminanten. Die Idee ist, dass ein System zur Mustererkennung anpassungsfähig aus Erfahrung lernt und

verschiedenartige Diskriminanten herausfiltert. Hier ist die Klassenzugehörigkeit von Interesse. Das System lernt aus den Beobachtungen der Muster, welche durch ihre Klasse identifiziert wird und folgert daraus eine Diskriminante zur Klassifikation. Doch zunächst noch ein paar einführende Worte zu den Feed-Forward-Netzwerken.

5.3 Feed-Forward-Netzwerke

Aus Erfahrung hat sich gezeigt, dass sich das Problem der Mustererkennung mit Hilfe eines Perceptron-Netzwerkes oder eines Feed-Forward-Netzwerks (auch Backpropagation-Netzwerk genannt) lösen lässt. Voraustests zu dieser Problematik haben diesen Lösungsansatz bestätigt.

Ein Backpropagation-Netzwerk besteht aus einer Eingabeschicht (input layer), einer Ausgabeschicht (output layer) und beliebig vielen verdeckten Schichten (hidden layer). Die Verbindungen laufen immer von unten (Eingabeseite) nach oben (Ausgabeseite), wobei nicht nur direkt benachbarte Schichten Verbindungen haben dürfen: Eine Verbindung kann auch beliebig viele Zwischenschichten überspringen. Werden Rückkopplungen eingebaut (recurrent connections) oder State Units wie in Jordan- oder Elman-Netzwerken, so verliert das Netzwerk seinen Feed-Forward-Charakter.

Eine lineare Diskriminante kann schematisch in Form einer Anordnung oder Array von Multiplikatoren und Summierungen repräsentiert werden. Der erste Schritt der Verarbeitung der Information der Eingangswerte besteht aus der Multiplikation aller Eingänge i_n mit dem zugehörigem Gewicht *w(ki)* und einer anschließender Summation. Der Eingang zu einem Element (Node) der nächsten Ebene errechnet sich dann wie folgt:

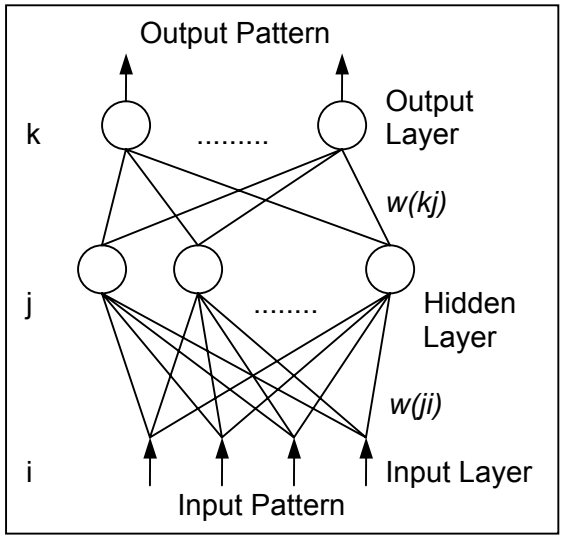


Abbildung 5.3: schematischer Aufbau eines Netzes

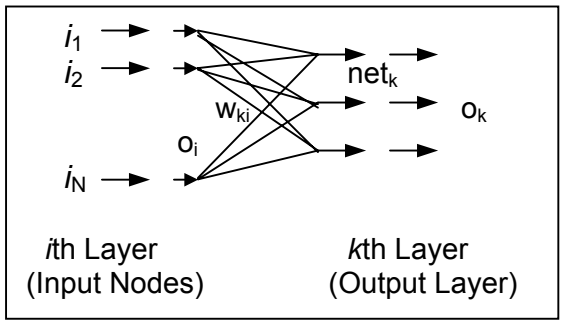


Abbildung 5.4: Input Layer – Output Probagation

$$net_j = \sum w_{ji} o_{ji} \qquad (5.1)$$

Der Ausgang des Nodes *j* ist dann: $o_j = f(net_j)$ (5.2)

Wobei f die Aktivierungs- oder Transferfunktion für den Node j ist. Diese Transferfunktion der Nodes muss differenzierbar sein. Die Gründe hierfür liegen im Lernen nach der generalisierten Delta-Lernregel. Unter dieser Voraussetzung kann jede beliebige Transferfunktion Verwendung finden. Darüber hinaus kann für jeden Node eine andere Transferfunktion verwendet werden. Der Einfachheit halber wählt man allgemeinhin zumindest für jedes Layer eine einheitliche Transferfunktion. Die am häufigsten eingesetzte Funktion im Backpropagation-Netzwerk ist die Sigmoid-Funktion. Für den Ausgang des

Nodes o_j ergibt sich dann:

$$o_j = \frac{1}{1 + e^{-(net_j + \theta_j)/\theta_0}} \qquad (5.3)$$

Der Parameter θ_j dient als Schwellwert. Ist dieser positiv so verschiebt sich die Transferfunktion nach links entlang der horizontalen Achse. Der Parameter θ_0 lässt das Aussehen der Funktion im Verlauf flacher oder steiler erscheinen.

Der sigmoidale Kurvenverlauf entspricht häufig der Beschreibung von natürlichen, belebten Vorgängen und bietet verschiedene Vorteile:

- jeder beliebige Eingangswert wird auf das Intervall zwischen null und eins abgebildet
- Eingangswerte, die nahe bei Null liegen, können durch die Steilheit der Kurve in diesem Bereich stärker auseinandergezogen und getrennt werden
- sehr große positive oder negative Werte führen immer zu Aktivitäten nahe Eins respektive nahe Null

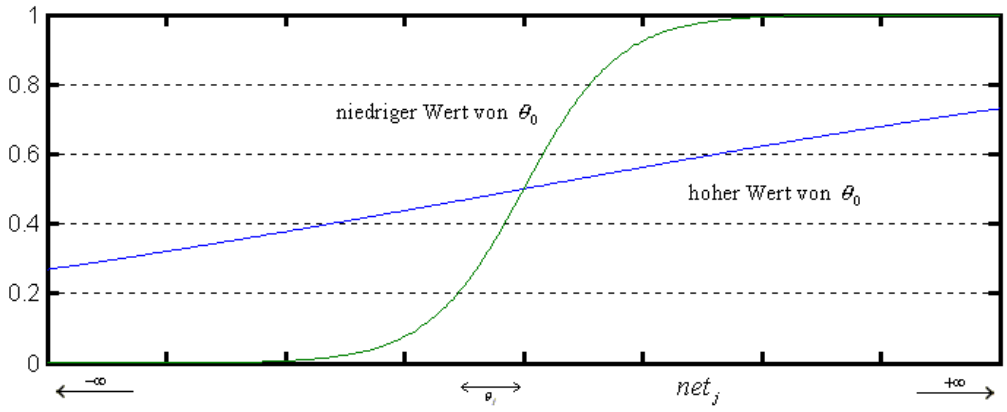


Abbildung 5.5: Bei der Berechnung der schnell steigenden Funktionskurve (grün) wurde $\theta_0 = 1$ gesetzt, bei der langsamer steigenden Kurve (blau) ist $\theta_0 = 10$ gesetzt worden.

Die ursprünglichen linearen Aktivierungsfunktionen sind eher unzulänglich und ermöglichen keine gute Annäherung an die Lösung. Sie haben stets die Form $o = D * net$. Weitere, relativ häufig eingesetzte Transferfunktionen sind die Sinusfunktion ($o = \sin(net)$), die Gaußfunktion ($o = e^{-net^2}$) und die Funktion des Tangens hyperbolicus ($o = \frac{2}{1 + e^{-2*net}} - 1$).

5.4 Extrahierung der Merkmale

Der Schriftsatz, welcher für den Net Monitor verwendet wird, liegt in Form einer kodierten Datei vor. Diese Muster können direkt ausgelesen und verarbeitet werden. Alle Informationen wie Höhe und Breite der Zeichen sind hierin enthalten und bedürfen keinen weiteren externen Angaben. In Abbildung 5.6 ist ein Ausschnitt aus der Fontdatei mit dem Zeichen „M“ zu sehen. (Hinter der Angabe GLYPH verbirgt sich der ASCII-Code des Zeichens.)

```
…
  GLYPH      0x004d    77
    WIDTH    0x0008     8
    HEIGHT   0x000c    12
    BASELINE 0x0008     8
    LINE  0 "##---##-"
    LINE  1 "##---##-"
    LINE  2 "#-#-#-#-"
    LINE  3 "#-#-#-#-"
    LINE  4 "#--#--#-"
    LINE  5 "#--#--#-"
    LINE  6 "#-----#-"
    LINE  7 "#-----#-"
    LINE  8 "#-----#-"
    LINE  9 "#-----#-"
    LINE 10 "--------"
    LINE 11 "--------"
  ENDGLYPH
…
```

Abb. 5.6: das Zeichen M aus dem Schriftsatz

Die Höhen- und Breitenangaben (WIDTH & HEIGHT) können aus zwei verschiedenen Gesichtspunkten betrachtet werden. Aus Perspektive von DisplayInspect stehen sie für Pixelelemente in Bezug auf das Werkzeug PixelProbe. Aus der Sicht der Mustererkennung sind dies Merkmale einer Klasse; hier der Klasse „M“. Die Motivation besteht nun darin, ausgewählte Zeichen des Schriftsatzes von einem Computer über ein neuronales Netzwerk erkennen zu lassen.

Im Anfangstadium der Erkennung gab es Probleme mit dem Programm DisplayInspect und dem PixelProbe-Werkzeug. Wie in den Abbildungen 5.12 zu sehen ist, werden die Zeichen meist originalgetreu, jedoch teils sporadisch verschoben um einen Pixelpunkt aus den Fehlerpixeln von PixelProbe extrahiert.

Nehmen wir an, dass wir die Schriftzeichen „0“, „1“, „2“, ... , „9“ aus dem Schriftsatz erkennen möchten. Dann haben wir die Situation, in der wir ein zu testendes Muster zur entsprechenden Klasse zuordnen müssen: das Muster des Schriftzeichens „0“ muss also der Klasse **0** zugeordnet werden, „1“ der Klasse **1** usw. D.h. wir müssen für jedes vorgelegte Muster eine Abschätzung durch ein Netz vornehmen lassen, zu welcher Klasse dieses Muster am besten gehören könnte. Genau in der Art wie unsere eigene Wahrnehmung funktioniert. Dieses Problem soll nun durch ein Backpropagation-Netzwerk gelöst werden.

Die Merkmale des Musters werden auf die Eingabeschicht (Input Pattern) übertragen. D.h. die Anzahl der Eingangselemente (Input Nodes) hängt vom jeweiligen Muster ab. Der hier verwendete Schriftsatz besitzt eine Größe von 96 Merkmalen oder Pixelpunkten und daraus ergibt sich die Größe der Eingangsschicht. Die Nodes in der Eingangsschicht werden mit positiven Werten für schwarze Punkte (im Muster mit „#“ gekennzeichnet) und negativen für die weißen besetzt. Die Größe der Ausgangsschicht wird bestimmt aus der Anzahl der zu klassifizierenden Muster. In unserem Falle sind dies 10, nämlich die Ziffern 0-9.

Im zweiten Schritt werden diese Werte über die Abbildungsfunktion des Netzwerks ausgeführt; dies entspricht dem Recall. Diese Funktion schließlich ist grundlegend für den daraus folgenden Ablauf des Wettbewerbs. Dasjenige Ausgangselement, welches am stärksten aktiviert wurde, zeigt mit größter Wahrscheinlichkeit die zugehörige Klasse des Musters an. Dieser Wettbewerb wird auch als „the-winner-takes-it-all“ bezeichnet.

5.5 Backpropagation oder Lernen mit der generalisierten Delta-Lernregel

Während der Lernphase eines solchen Netzwerkes wird ihm am Eingang ein Muster $\underline{x}_p = \{i_{pi}\}$ präsentiert und gefordert, dass das Netz alle Gewichte in allen Verbindungen und ebenso alle Schwellwerte in den Nodes justiert, sodass die gewünschten Ausgänge t_{pk} in allen errechneten Ausgangsnodes erreicht werden. Ist diese Prozedur beendet, werden dem Netz weitere $\underline{x}_p$'s und t_{pk}'s präsentiert und verlangt, dass das Netz auch diese einlernt. Es wird also gefordert, dass das Netz diejenigen Gewichte und Schwellwerte findet, welche zu allen Eingangs- und Ausgangsmustern passen, welches es präsentiert bekommt.

Im allgemeinen sind die Ausgänge $\{o_{pk}\}$ nicht die gleichen wie die gewünschten t_{pk}'s. Für jedes präsentierte Muster errechnet sich der quadratische Fehler nach

$$E_p = \frac{1}{2}\sum_k (t_{pk} - o_{pk})^2 \tag{5.4}$$

und der durchschnittliche Systemfehler nach

$$E = \frac{1}{2P}\sum_p \sum_k (t_{pk} - o_{pk})^2 \tag{5.5}$$

wobei der Faktor ½ lediglich wegen der einfacheren Herleitung der Lernregel eingesetzt wird.

Die generalisierte Delta-Lernregel, welche durch Rumelhart, Hinton und Williams [1986] formuliert wurde, zum Einlernen der Gewichte und Schwellwerte soll den Fehler E_p so schnell wie möglich in Richtung des steilsten Gradienten minimieren. Unterschiedliche Ergebnisse werden erzielt, ob die Suche auf Basis von E_p oder E basiert. Die Suche nach dem wahren Gradienten des minimalsten Systemfehlers sollte auf der Minimierung von E basieren. In der später noch vorgestellten Implementierung wurden jedoch beide Minimierungen beachtet. Unterschreiten alle E_p oder nur E den eingestellten Maximalwert, so gilt das Netzwerk als eingelernt.

Verbesserte Werte der Gewichte und Schwellwerte werden erreicht, wenn die Gewichtsänderung Δw_{ki} proportional zu $-\partial E / \partial w_{ki}$ erfolgt. Daraus folgt:

$$\Delta w_{kj} = -\eta \frac{\partial E}{\partial w_{kj}} \tag{5.6}$$

Die partielle Ableitung kann weiter mittels Kettenregel aufgelöst werden:

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} \tag{5.7}$$

Aus 5.3 folgt dann

$$\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}}\sum w_{kj} o_j = o_j \tag{5.8}$$

Definiert man nun $$\delta_k = -\frac{\partial E}{\partial net_k} \tag{5.9}$$

und kann schreiben $$\Delta w_{kj} = \eta \delta_k o_j \tag{5.10}$$

Um $\delta_k = -\partial E / \partial net_k$ zu errechnen, kann die Kettenregel angewendet werden. Die beiden Faktoren der partiellen Ableitungen drücken die Änderungsrate des Fehlers mit Blick auf den Ausgang o_k und die Änderungsrate des Ausgangs des Nodes k mit Blick auf den Eingang

desselben Nodes aus: $$\delta_k = -\frac{\partial E}{\partial net_k} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial net_k} \tag{5.11}$$

Für die beiden Faktoren kann man schreiben

$$\frac{\partial E}{\partial o_k} = -(t_k - o_k) \tag{5.12}$$

und $$\frac{\partial o_k}{\partial net_k} = f_k^{/}(net_k) \tag{5.13}$$

Hieraus folgt $$\delta_k = (t_k - o_k) f_k^{/}(net_k) \tag{5.14}$$

für jeden Ausgangslayernode k, und daraus folgt

$$\Delta w_{kj} = \eta (t_k - o_k) f_k^{/}(net_k) o_j = \eta \delta_k o_j \tag{5.15}$$

Wenn die Gewichte nicht direkt die Ausgangsnodes beeinflussen, kann man immer noch schreiben

$$\begin{aligned}
\Delta w_{ji} &= -\eta \frac{\partial E}{\partial w_{ji}} \\
&= -\eta \frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} \\
&= -\eta \frac{\partial E}{\partial net_j} o_i \\
&= \eta \left(-\frac{\partial E}{\partial o_j}\frac{\partial o_j}{\partial net_j} \right) o_j = \eta \left(-\frac{\partial E}{\partial o_j} \right) f_j^{/}(net_j) o_j \\
&= \eta \delta_j o_j
\end{aligned} \tag{5.16}$$

Der Faktor $\partial E / \partial o_j$ kann nicht direkt gelöst werden. Stattdessen wird er durch bekannte und lösbare Werte aufgelöst:

$$\begin{aligned}
\frac{\partial E}{\partial o_j} &= -\sum_k \frac{\partial E}{\partial net_k}\frac{\partial net_k}{\partial o_j} = \sum_k \left(-\frac{\partial E}{\partial net_k} \right) \frac{\partial}{\partial o_j} \sum_m w_{km} o_m \\
&= \sum_k \left(-\frac{\partial E}{\partial net_k} \right) w_{kj} = \sum_k \delta_k w_{kj}
\end{aligned} \tag{5.17}$$

In diesem Falle ist $$\delta_j = f_j'(net_j)\sum_k \delta_k w_{kj} \tag{5.18}$$

Das bedeutet, dass die Deltas eines internen Nodes aus den Deltas der Nodes einer höheren Schicht berechnet werden können. Ausgehend von der höchsten Schicht – der Ausgangsschicht – lässt sich δ_k aus (5.14) berechnen und dieser Fehler wird zurückpropagiert und zur Berechnung der Fehler der vorhergehenden Schicht verwandt. Wird ein zusätzlicher Bezeichner p für ein Muster (Pattern) benutzt, dann ergibt dies

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi} \tag{5.19}$$

Falls j Nodes einer Ausgangsschicht sind, gibt dies

$$\delta_{pj} = (t_{pj} - o_{pj}) f_j'(net_{pj}) \tag{5.20}$$

Falls jedoch j Nodes einer verdeckten Schicht sind, dann müssen zuvor die δ s der

höheren Schicht berechnet werden: $$\delta_{pj} = f_j'(net_{pj})\sum_k \delta_{pk} w_{kj} \tag{5.21}$$

Im Speziellen, falls $$o_j = \frac{1}{1+\exp\left[-\left(\sum_i w_{ji} o_i + \theta_j\right)\right]} \tag{5.22}$$

dann ist $$\frac{\partial o_j}{\partial net_j} = o_j(1-o_j) \tag{5.23}$$

und die Deltas sind gegeben $$\delta_{pk} = (t_{pk} - o_{pk})\, o_{pk}(1-o_{pk}) \tag{5.24}$$

$$\delta_{pj} = o_{pj}(1-o_{pj})\sum_k \delta_{pk} w_{kj} \tag{5.25}$$

für die Ausgangsschicht und respektive für die verdeckte Schicht.

Die Schwellwerte θ_j werden auf gleiche Weise wie die anderen Gewichte eingelernt. θ_j kann als Gewicht eines Nodes mit einem unendlichen Ausgangswert angesehen werden.

Die Initialisierung aller Gewichte erfolgt durch zufällige Werte (hier: zwischen –0.5 und 0.5). Der Grund hierfür ist die entstehende Symmetrie in den verdeckten Schichten. Würden alle Nodes einer höheren Schicht die gleichen Eingangswerte erhalten, haben auch alle Nodes die gleiche Aktivität. Da auch der Fehler von der nachgeschalteten Schicht für alle Nodes mit gleicher Aktivität identisch ist, berechnen sie genau die gleichen Gewichtsänderungen. Dadurch kann das Netzwerk nicht konvergieren und unterschiedliche Klassen repräsentieren.

Die Lernprozedur besteht aus der Initialisierung der Gewichte, der Präsentation eines Übungsmusters am Eingang des Netzes, der Berechnung der Ausgänge im Feed-Forward-Verfahren. Der Fehler der Ausgänge wird anfangs etwas groß sein, wodurch die Gewichte

verändert werden müssen. Über das Verfahren der Backpropagation errechnet das Netz nun $\Delta_p w_{ji}$ für alle w_{ji} für das angelegte Muster p. Die Prozedur wird nun für alle Muster der Klasse wiederholt. Die Korrektur der Gewichte wird angestoßen und die Ausgänge werden erneut nach dem Feed-Forward-Verfahren berechnet. Die Unterschiede zwischen aktuellem und gewolltem Ausgang führen wiederum zu Gewichtsänderungen. In einem erfolgreichem Lernzyklus wird der Systemfehler mit der Anzahl der Iterationen verringert.

Rumelhart, Hinton und Williams schlagen zudem vor, eine Art Momentum Rate α der Gleichungen (5.15) und (5.18) hinzuzufügen und damit lautet die Lernregel:

$$\Delta w_{ji}(n+1) = \eta \delta_j o_j + \alpha \Delta w_{ji}(n) \tag{5.26}$$

Die Einführung des Momentums kann weitgehend die oszillierenden Eigenschaften dämpfen, jedoch nachteilig auch langsamere Lernraten verursachen. In der Berechnung des neuen Gewichts zum Zeitpunkt $n+1$ geht dann zusätzlich die Gewichtsänderung des vorangegangenen Lernschrittes zum Zeitpunkt n und damit auch der alte Fehler ein. Diese Methode ist dann keine echte Gradientensuche mehr.

Zusätzlich sollte der Wert η klein gewählt werden, weil es sonst große Änderungen der Gewichte nach sich ziehen kann. Wird ein großer Wert gewählt, so kann andererseits schneller gelernt werden. In Versuchen haben sich Werte zwischen 0.4 und 0.9 als ausreichend herausgestellt. Die Regel lautet: Je kleiner η, desto geringer die Konvergenz und desto minimaler die Wahrscheinlichkeit großer Oszillationen an den Ausgängen.

5.6 Einflussfaktoren auf das Lernen

Da das Backpropagation-Netzwerk immer noch eines der am häufigsten eingesetzten neuronalen Netze ist, unterliegt dies auch ständigen Versuchen die Lernregeln zu verbessern und zu beschleunigen. Einige wichtige und grundsätzliche Änderungen werden nun vorgestellt.

Die einfachste Möglichkeit bietet sich mit Hilfe der **Skalierung** bei der Präsentation der Muster und gewünschten Ausgänge der Klassen, welches das Netzwerk einlernen soll. Ein weitverbreiteter Glaube herrscht dahingehend, dass die Eingangsmuster zwischen den Werten 0 und 1 skaliert werden müssen. Dies ist jedoch nicht unbedingt notwendig und gut. Für die Eingangswerte, welche auf den Wert 0 gesetzt sind, können die Gewichte nicht justiert werden. Diese nehmen somit nicht am Lernen teil. Werden die Muster auf die Werte zwischen -0.5 und 0.5 für weise respektive schwarze Punkte symmetrisch skaliert, so erhöht sich die Lerngeschwindigkeit auf fast magische Weise. Hier ist noch anzumerken, dass durch die Aktivierungsfunktion unter (5.22) ein Node relativ schlecht die Endwerte 1 oder 0 erreicht, es sei denn die Gewichte sind ziemlich groß oder klein. Im Lernprozess werden daher Ausgangswerte für gesuchte Klassenzugehörigkeit von 0.9 und keine Zugehörigkeit von 0.1 als ausreichend betrachtet, um binäre Aussagen zu treffen.

Eine weitere Möglichkeit besteht in der Optimierung der Konvergenzgeschwindigkeit im **Fastback**-Verfahren. Diese Variante der Lernregel wurde 1988 von Tarig Samad eingeführt: Für die Anpassung des Gewichtes w_{ji} wird der Fehler des Nodes i auf seinen Ausgangswert addiert. Anstatt des tatsächlichen Ausgangswertes o_i wird der Erwartungswert $(o_i + \delta_i)$ für die Gewichtsanpassung verwendet. Damit wird aus (5.26):

$$\Delta w_{ji}(n+1) = \eta \delta_j (o_j + k\delta_i) + \alpha \Delta w_{ji}(n) \qquad (5.27)$$

Für $k = 1$ wird genau der Erwartungswert der Ursprungseinheit verwendet. Es wird empfohlen für k: $1.0 \le k \le 1{,}5$ zu setzen.

Des weiteren bietet sich bei der **Justierung der Steilheit** der Kurve eine Möglichkeit. Wenn die Steilheit der Kurve erhöht wird, verlangsamt sich die Lernzeit. In einem Artikel von Izui und Pentland wird auf die fortführende Mathematik dieser Thematik eingegangen.

Durch **direkte Verbindungen** zwischen Eingangs- und Ausgangsschichten können sich zusätzlich die Lernraten ändern. Diese Methode arbeitet gut, wenn die zu findende Funktion nahezu linear ist. Die Anzahl der Nodes in versteckten Schichten kann zudem reduziert werden. Diese Methode ist jedoch nicht angebracht, wenn es viele Ausgänge im Netzwerk gibt.

Unter den noch so vielen Verbesserungsmöglichkeiten sei noch eine letzte hier erwähnt, nämlich die Methode der **dynamischen Anpassung der Lernrate**. Wird die Lernrate ständig in den optimalen Bereich gestellt, sollte die Lerngeschwindigkeit zunehmen und gleichzeitig sollte die Zuverlässigkeit der Konvergenz steigen. Die hierfür verwendeten Verfahren sind vielfältig. Das Problem besteht in der korrekten Wahl der Lernrate zum richtigen Zeitpunkt. Die Vielfalt dieser Verfahren hier noch auszuarbeiten, würde den Rahmen sprengen. Daher möchte ich hier keine weiteren Angaben einer bestimmten Lösung machen.

5.7 Anmerkung zu versteckten Schichten

In der Theorie wird behauptet, dass für die meisten nichtpolynomischen Funktionen mit linearen Ausgangselementen eine Lage verdeckter Schichten ausreicht. Die Elemente in der verdeckten Schicht sollten $N-1$ betragen, wobei N die Anzahl der Muster sei. Andere Behauptungen raten, etwa so viele Elemente in der verdeckten Schicht zu haben wie Lernmuster vorhanden sind. Und viele andere Leute stellten noch mehr Formeln zur Findung der optimalen Anzahl der Elemente auf, jedoch konnte sich keine einzige als wirklich vorteilhaft beweisen. Die beste Methode zur Findung ist immer die des Versuchs und Irrtums. Andererseits ist es vielleicht ratsamer, mit so wenig wie möglich Elementen in der verdeckten Schicht auszukommen und falls notwendig die Anzahl der Elemente zu vergrößern. Dies kann die Rechenzeit u.a. verringern. Eine weitere Frage besteht darin, ob ein, zwei oder mehr verdeckte Schichten am besten wären. Hier sei auch wieder erwähnt, dass es dafür kein Patentrezept gibt.

Im selbstentwickelten Programm NNUI können verschiedene Feed-Forward-Netzwerke durch freie Parametrierung auf Ihre Tauglichkeit untersucht werden, wie im nächsten Kapitel gezeigt werden soll. Hier hat sich herausgestellt, dass unter Umständen eine höhere Anzahl verdeckter Elemente wie Eingangselemente sich besser zur Mustererkennung eignen.

5.8 NNUI – die Optionen

Zunächst ein paar Worte zu den Optionen. Durch freie Parametrierung von α, η, den maximalen Werten vom Systemfehler E und vom quadratischen Fehler E_p, der Größe und Anzahl verdeckter Schichten sowie deren Elemente, bis hin zu den maximalen Lernepochen und der Anzahl der Lernwiederholungen können Netzwerke zur Mustererkennung trainiert und anschließend auf Ihre Tauglichkeit untersucht werden.

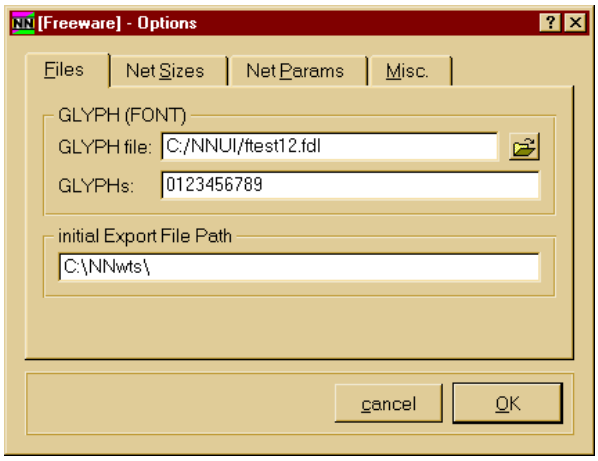


Abbildung 5.7: Dateieinstellungen für die Fontdatei, die zu erkennenden Zeichen, und den Pfad für zu exportierende Netze

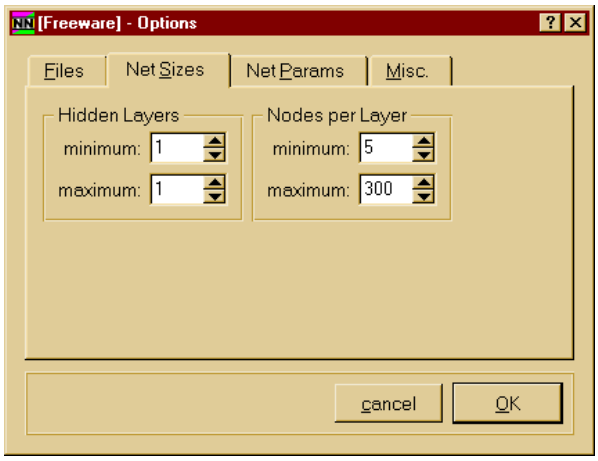


Abbildung 5.8: Einstellungen für minimale und maximale Werte zur Berechnung für versteckte Schichten und deren Elemente

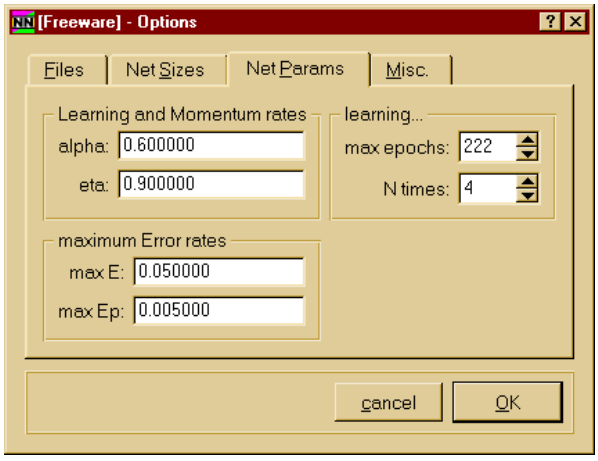


Abbildung 5.9: Netzwerkparameter für alle zu errechnenden Netze: α, η, E und E_p, sowie die maximale Anzahl der Lernepochen und Lernwiederholungen

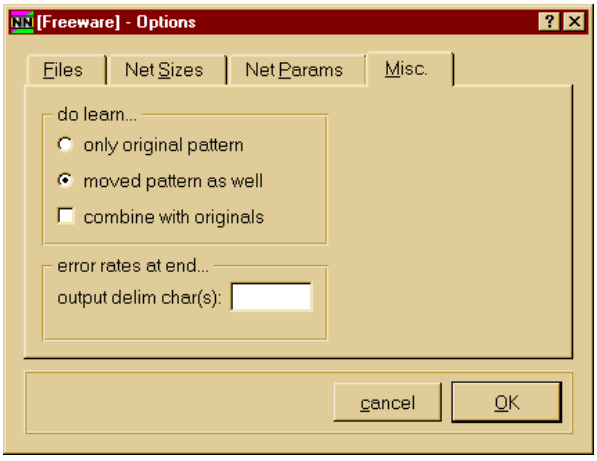


Abbildung 5.10: do learn: Originale, Verschiebungen und beide in Kombination der Muster können eingelernt werden; Trennzeichen für die Liste aller Fehlerraten

In Abbildung 5.7 sind die Einstellungen zu den Dateien und Verzeichnissen zu sehen. Die Schriftsatzdatei wird mit dem Parameter „Glyph File“ eingestellt. In dieser Datei müssen die Schriftzeichen wie in Abbildung 5.6 kodiert sein. Unter der Einstellung „Glyphs“ sind die Zeichen einzutragen, welche eingelernt werden sollen. Im Optionsfeld „Export File Path“ sollte das Verzeichnis eingetragen werden, in welches die ausgewählten errechneten Netzwerke exportiert werden sollen. In der Hauptansicht des Programms (Abbildung 5.11) können die Netzwerke aus der Listbox (linke Seite) zum Export markiert werden, und mit dem Knopf „export weights“ werden diese Netzwerke in das angegebene Verzeichnis als Dateien exportiert. Die Dateinamen werden aus den zu exportierenden, gelisteten Netzwerkgrößenangaben generiert. In jeder Datei stehen die Daten des Netzwerks wie die zu klassifizierenden Zeichen, die Anzahl und Größen der Schichten und die Gewichte des Netzwerks.

Die Größen der Ein- und Ausgangsschichten sind durch den Schriftsatz und die Anzahl der einzulernenden und zu erkennenden Zeichen fest vorgegeben. Wie schon im Kapitel (5.4) erwähnt, ist die Größe der Eingangsschicht abhängig von der Pixelgröße der Zeichen (oder Muster). Die Größe der Ausgangsschicht richtet sich nach der Anzahl der zu erkennenden Zeichen, also der Anzahl der vorzunehmenden Klassifikationen (siehe Optionsfeld „Glyphs“). Die minimale und maximale Anzahl der verdeckten Schichten sowie die maximale und minimale Anzahl der Elemente einer Schicht können, wie in Abbildung 5.8 zu sehen, eingestellt werden.

In den Einstellungen „Net Params“ unter Abbildung 5.9 können die Parameter α, η, E und E_p sowie die maximale Anzahl der Lernepochen und die Anzahl der Lernwiederholungen eingestellt werden. Diese Angaben sind für alle einzulernenden Netzwerke gültig.

In Abbildung 5.10 kann unter „do learn“ eingestellt werden, ob nur die Originalmuster oder auch deren Verschiebungen (siehe Abbildung 5.13) eingelernt werden sollen. Wenn die Einstellung der Verschiebung gewählt wurde können die Originale und ihre Verschiebungen überlagert eingelernt werden.

Am Ende der Berechnung werden alle Fehlerraten noch einmal zusammengefasst und hintereinander durch ein Trennzeichen getrennt im Hauptfenster ausgegeben. Dieses Trennzeichen kann mit dem „Delimiting Character“ angepasst werden.

Nach diesen Angaben eingelernte und exportierte Netzwerke können nun externe Programme ihre eigenen Netzwerke aufbauen und schließlich Muster klassifizieren.

5.9 NNUI – das Programm zum Einlernen

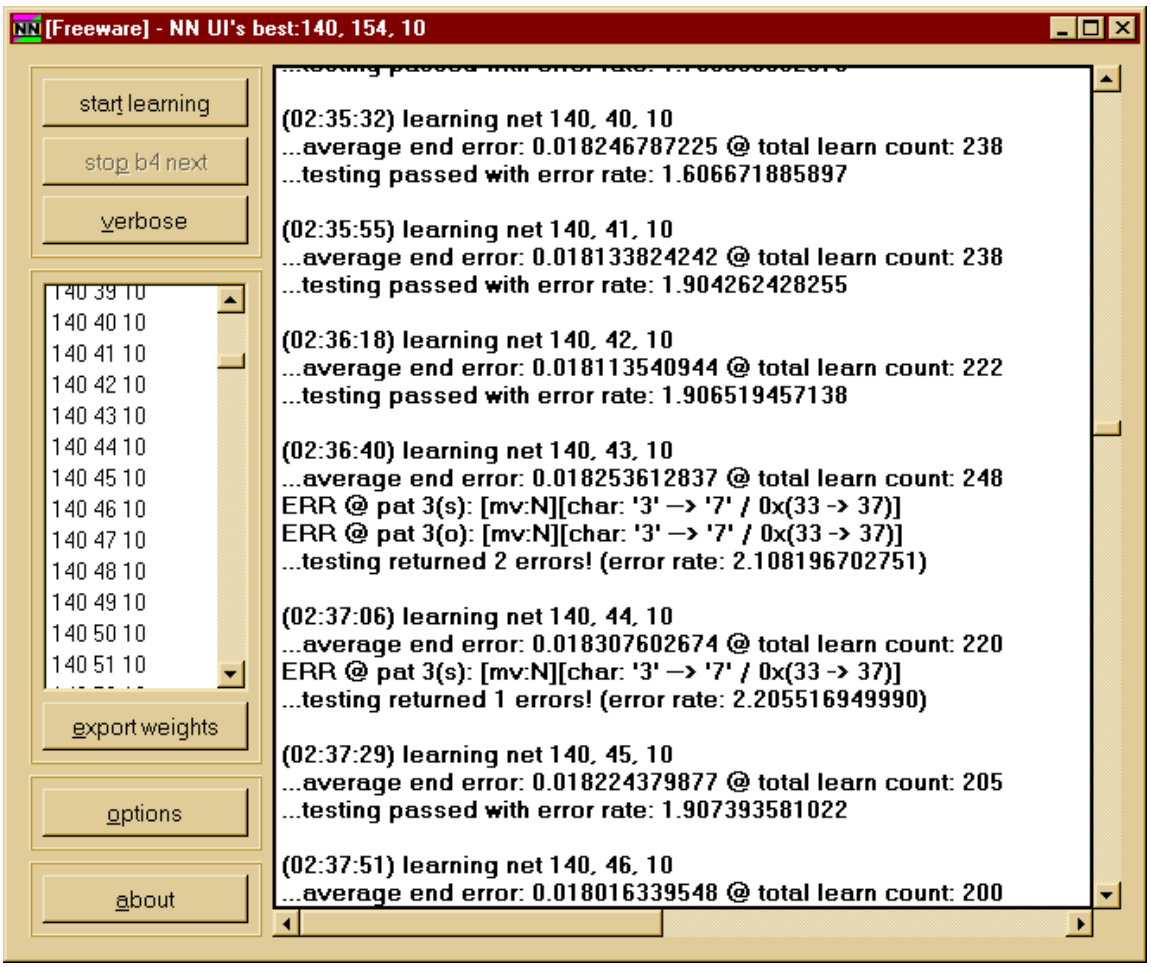


Abbildung 5.11: NNUI bei der Berechnung eines Netzes nach den Optionen aus Abb. 5.7-5.10.

Im großen, fortlaufenden Statusfenster von NNUI kann der Fortschritt der Errechnung der Netzwerke und Gewichte verfolgt werden. Ein besonderes Merkmal des Statusfensters ist, dass es editierbar und damit ebenfalls exportierbar ist.

Nachfolgend ein paar Erläuterungen zu den Statusausgaben:

(02:37:06) learning net 140, 44, 10

In dieser Zeile steht der Startzeitpunkt des Einlernens des Netzwerks mit 140 Eingangs-, 29 versteckten und 10 Ausgangselementen (Nodes). Wird die Startzeit des nächsten Netzwerkes vom Zeitpunkt des Einlernens abgezogen, so ergibt sich in etwa die Dauer der Lernphase. Zum Zeitpunkt dieser Aufnahmen wurde das Programm auf einem Rechner mit AMD K6-II Prozessor, 200MHz und 96MB RAM unter MS-Windows 98 ausgeführt.

Während des Lernens erscheint eine Fortschrittsanzeige mit der Angabe der Lernwiederholung (hier: 1, das erste Einlernen). Die Punkte zeigen den Fortschritt innerhalb des Lernzyklus' der Verschiebung der Klassifikationsmuster (X1-X9) an. Dazu mehr im Kapitel 5.9.

learning 1: .../

Während der Lernphase wird der Gesamtsystemfehler E aufsummiert und am Ende der Berechnung geteilt durch die Anzahl der Muster und Lernwiederholungen. Dies ergibt letztlich den Durchschnitt des Fehlers. Diese Angabe ist nur beschränkt über die Güte eines Netzwerks aussagefähig. Ist dieser Wert jedoch zu groß, sollte das Netzwerk nicht zum Erkennen benutzt werden:

...average end error: 0.018307602674 @ total learn count: 220

Nach der Lernphase folgt eine kurze Testphase, in dem alle eingelernten Muster einmal auf korrekte Klassifikation abgetestet werden. Der Sinn dahinter ist, dass eingelernte Netze, deren Systemfehler selbst unter der eingestellten, maximalen Schwelle Probleme haben, das eine oder andere Muster korrekt zu erkennen. Die Ausgaben in Form von

ERR @ pat 3(s): [mv: N][char: '3' --> '7' / 0x(33 -> 37)]

geben über falsche Klassifikationen Auskunft. Hier konnte das 3. Muster mit einer Verschiebung ohne Überlagerung des Originals Richtung Nord nicht korrekt klassifiziert werden. Es erkannte eine ‘7’ anstatt einer ‘3’. Die letzten beiden Angaben repräsentieren diese Angaben in hexadezimaler Darstellung.

In diesem Netz wurde ein Muster nicht korrekt erkannt. Mit der letzten Angabe, der Fehlerrate, kann letztlich über die Güte des Netzwerks eine Aussage getroffen werden:

...testing returned 1 errors! (error rate: 2.2055169990)

Die Fehlerrate bestimmt sich aus der Summe der Unterschiede von vorgegebener zu errechneter Klassifikation der Ausgangsnodes. Die vorgegebenen Werte zur Klassifikation wurden auf die Werte 0.1 und 0.9 skaliert. Liegt der Unterschied von vorgegebenem zu errechnetem Ausgangswert höher als 0.1, so wird das Zehnfache des Unterschieds aufsummiert. Dies hat den Vorteil, eine etwas bessere Aussage über das Netz zu bekommen. Je kleiner diese Fehlerraten unter dem Wert Eins liegen, desto besser haben sie sich auch in der Praxis bewährt. In Kapitel 5.10 werden die Fehlerraten der Netzwerke graphisch aufbereitet.

Ist zusätzlich eine Statusausgabe mit folgendem Inhalt gegeben,

max epochs: [L1@SO,S,SW],

so konnten innerhalb der angezeigten Lernphase – L1: hier die erste – die Muster mit den Verschiebungen – hier nach SO, S und SW – bis zur maximalen Grenze der Lernepochen die quadratischen oder Systemfehler nicht unterschreiten.

Am Ende der gesamten Lernzeit für alle eingestellten Netze werden alle Fehlerraten und die gesamt benötigte Zeit ausgegeben. Beispielhaft für die Zeit:

learn duration: 5410 seconds || 90.18 minutes || 1,513 hours

5.10 Verbesserung durch Musterverschiebung

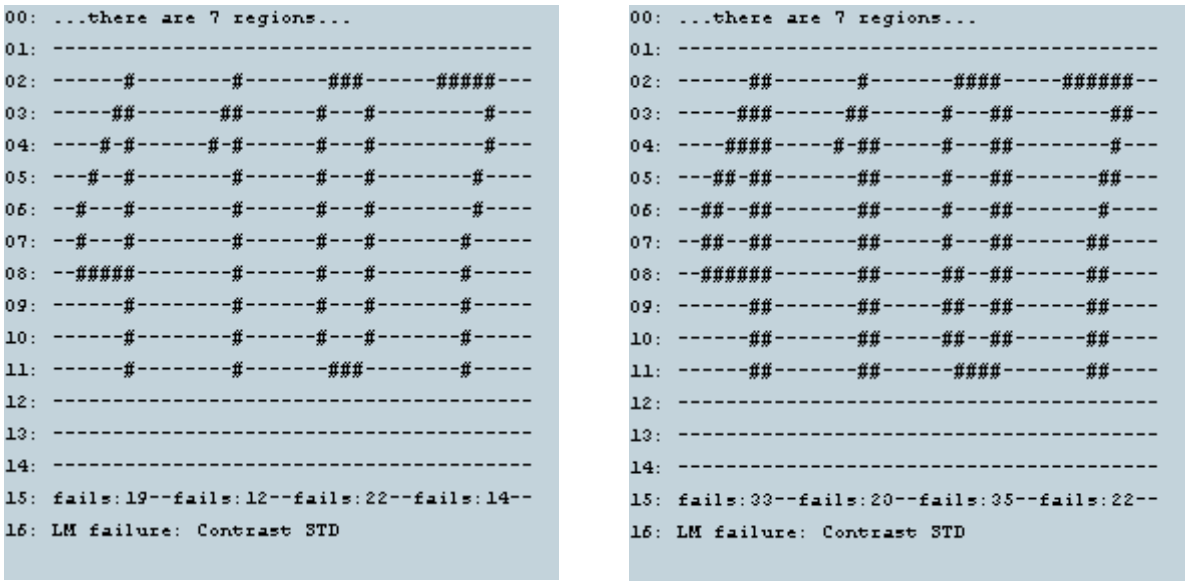


Abbildung 5.12: linkes Bild: korrekte Extrahierung der Pixel, rechte Seite: während längerer Testreihen ergeben sich sporadische Verschiebung von Pixelprobe

Wie schon in einem der vorigen Kapitel erwähnt, sollte das Netzwerk so eingelernt werden, dass es in der Lage ist, einen Teil des Schriftsatzes zu erkennen und die erkannte Klassenzugehörigkeit in geeigneter Form zurückzugeben.

Das Werkzeug PixelProbe des Programms DisplayInspect kann so ausgerichtet werden, dass es genau über dem zu analysierenden Zeichen steht. PixelProbe wird in ein Raster nach Höhe und Breite entsprechend dem darrunterliegenden Zeichen unterteilt. Bevor die Pixel herausgeholt werden können, muss das Tool mit einem leeren Bild eingelernt werden, weil nur die fehlerhaften Pixel erscheinen. Fehlerhaft sind Pixel dann, wenn sich ein Unterschied zwischen eingelerntem und getestetem Bild in Form von Intensität, Kontrast, Farbe oder An-Aus auswirkt. Die beiden Bilder in Abbildung 5.9 verdeutlichen dies. In Zeile 15 sind die jeweiligen Mengenangaben der Fehler je PixelProbe-Werkzeug angegeben.

Durch Bewegung und Ungenauigkeiten von Kamera zum Objekt können sich die Aufnahmen in alle Richtungen verschieben. Im rechten Bild von Abbildung 5.12 ist eine Verschiebung nach rechts zu sehen. Beim einfachem und sturem Einlernen der Muster nach Zeichensatzvorlage und nachfolgender Erkennung verschobener Resultate

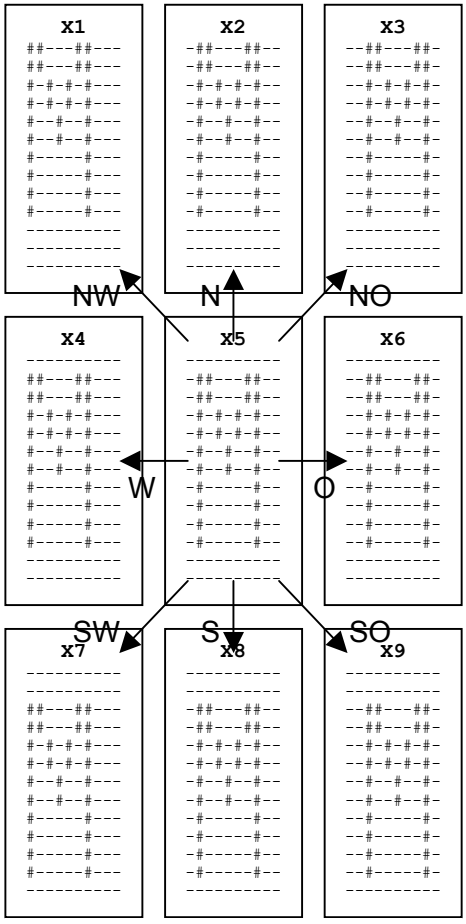


Abb. 5.13: Verschiebung der Muster während der Lernphase

kann dies Probleme falscher Klassifikationen nach sich ziehen. Zur Behebung dieser „Erkennungsfehler“ wurde das „moving“ Muster zum Einlernen verwendet, was bedeutet, dass ein Muster mehrfach verschoben um jeweils einen Pixelpunkt eingelernt wird. Dies ergibt neun Möglichkeiten ein Muster einzulernen (siehe Abbildung 5.10), jeweils verschoben nach N, NO, O, SO, S, SW, W, NW (nach Himmelsrichtungen) und M für die Mitte (Originalvorlage mit Rand). Für nochmalige Verbesserung sorgte das Einlernen von Verschiebung und Original gleichzeitig, also überlagert. Für diese Problematik müssen die Muster zum Einlernen und jedes der vier PixelProbe Elemente um einen Randpixel vergrößert oder erweitert werden, da sonst Merkmale durch Verschiebung verloren gehen können.

5.11 Ergebnisse

Die in diesem Kapitel werden die Graphen der Fehlerraten, unter Netzwerken mit einer verdeckten Schicht und einer angegebenen Klassifikation dargestellt. Wie schon in Kapitel 5.8 beschrieben, geben die Fehlerraten über die Erkennung der Lernmuster eine gewisse Güte an. Die Angaben zur Einlernzeit beruhen einer Prozessorgeschwindigkeit von 1,7GHz. Systemauslastung und wechselnde Prozessprioritäten nehmen noch einmal Einfluss darauf.

Allen Netzwerken zugrunde liegen die Parameter: $\alpha = 0.6$, $\eta = 0.9$, $E = 0.03$, $E_p = 0.003$, $\max \text{Epochs} = 222$ und $\text{Lernwiederholungen} = 4$.

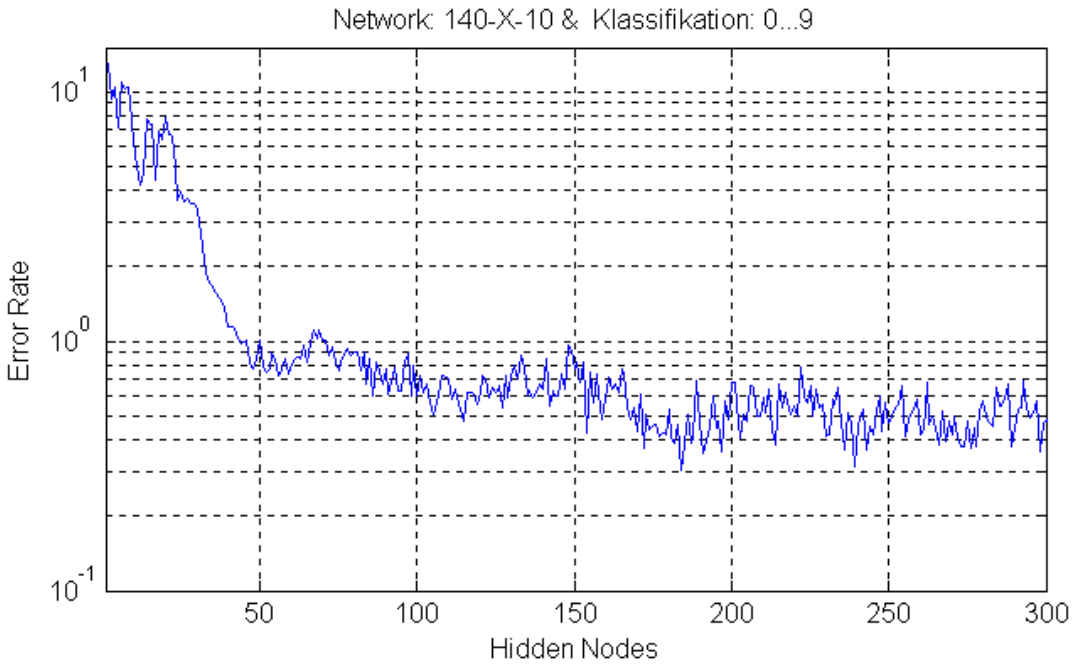


Abbildung 5.14: Fehlerraten des Netzwerks mit 140 Eingangs- und 10 Ausgangselementen und der Klassifikation der numerischen Ziffern 0-9, aufgetragen als Funktion der versteckten Elemente - in logarithmischer Darstellung. Der Tiefpunkt wurde mit x=183 versteckten Elementen mit einer Fehlerrate von ca. 0.3053 erreicht. Die Einlernzeit betrug ca. 1,5 Stunden.

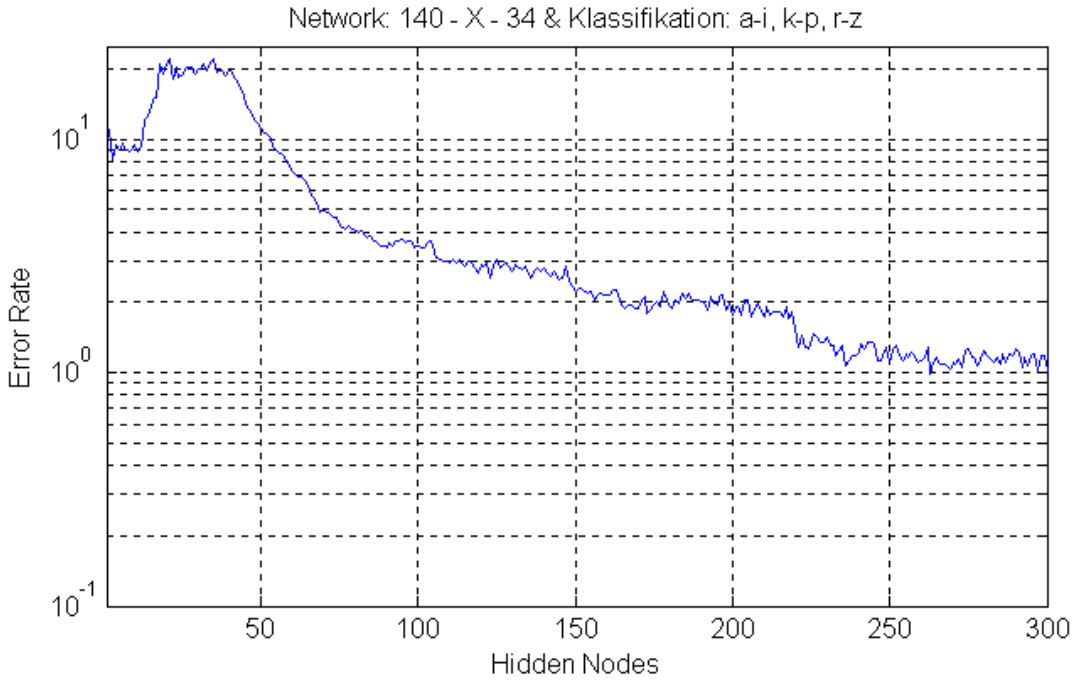


Abbildung 5.15: Fehlerraten des Netzwerks mit 140 Eingangs- und 24 Ausgangselementen und der Klassifikation der kleinen alphabetischen Zeichen a-i, k-p, r-z, aufgetragen als Funktion der versteckten Elemente - in logarithmischer Darstellung. Der Tiefpunkt wurde bei x=262 mit ca. 0.9834 erreicht. Die Einlernzeit betrug ca. 7 Stunden. Hier ist gut zu erkennen, dass mindestens ein Zeichen mit seiner Verschiebung einem anderen ähneln muss. Die Lernrate sinkt nicht wirklich unter den Wert Eins.

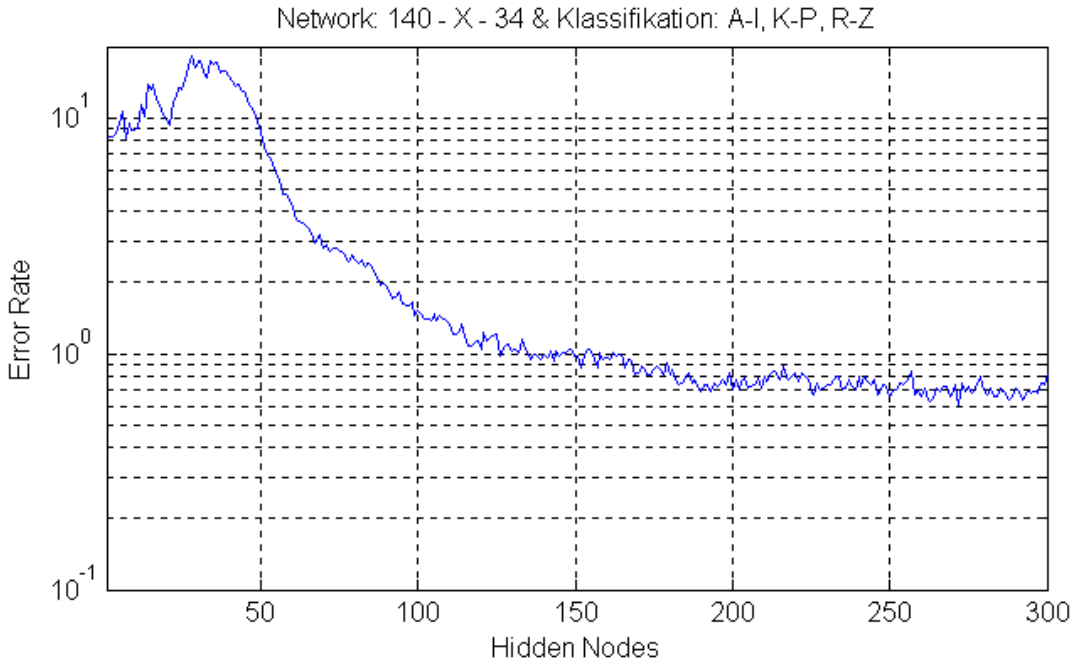


Abbildung 5.16: Fehlerraten des Netzwerks mit 140 Eingangs- und 24 Ausgangselementen und der Klassifikation der großen alphabetischen Zeichen A-I, K-P, R-Z, aufgetragen als Funktion der versteckten Elemente - in logarithmischer Darstellung. Der Tiefpunkt wurde bei x=271 mit ca. 0.6030 erreicht. Die Einlernzeit betrug ca. 8 Stunden.

6 AUDIOVERGLEICH

Der Befehl SNDCMP – von Sound Compare – soll die Aufgabe haben, zwei Audioaufnahmen, also Töne oder ganze Musikstücke, vergleichen zu können. Die Idee dabei ist die, dass wenn Bilder verglichen werden können, könnte man auch zwei Audiosequenzen vergleichen oder gegenüberstellen. Diese Problematik der Findung von Übereinstimmungen von zwei Audiosignalen soll in diesem Kapitel erörtert werden.

Audiosignale sind zeitlich variable Messgrößen. Diese bestehen meist aus periodischen und nicht periodischen Signalen und Rauschen. Mit der Fouriertransformation können die verschiedenen Informationen des Signals wie das Spektrum, die Dichte und die Korrelation leicht errechnet werden. Meist sind die Algorithmen rund um die Fouriertransformation in den Mathematikprogrammen integriert und vollständig implementiert, sodass sie nur noch angewendet werden müssen, so muss man sich über die erwarteten Ergebnisse im klaren sein. In den ersten vier Unterkapiteln werden deshalb die Grundlagen der verwendeten Mathematik angerissen. Diese erstrecken sich von kontinuierlicher und diskreter Fouriertransformation über Korrelation bis hin zum angewendeten Lösungsansatz, welcher im Abschluss des Kapitels gegeben wird.

6.1 Kontinuierliche Fouriertransformation

Fourierreihen ermöglichen es, eine periodische Funktion mit Frequenzen der Vielfachen der Grundfrequenz als unendliche Summe harmonischer Schwingungen darzustellen. Die Fouriertransformation andererseits ermöglicht es, eine nichtperiodische Funktion als Integralsumme von fortlaufenden, ununterbrochenen Serien von Frequenzen darzustellen. Die Fourierreihen werden manchmal als Spezialfall der Fouriertransformation betrachtet.

Ein physikalischer Prozess kann entweder in der Zeitdomäne, von den Werten f als Funktion der Zeit t, oder in der Frequenzdomäne als Integralsumme harmonischer Schwingungen beschrieben werden. Die bekannten, äquivalenten Gleichungen für die Fourierintegrale sind in den Gleichungen (6.1) – (6.6) zu sehen. Es ist nur eine Frage des Geschmacks oder mathematischen Konvention, welche der folgenden Gleichungen angebracht ist.

$$f(t) = \int_0^\infty \{A(\omega)\cos\omega t + B(\omega)\sin\omega t\} d\omega \qquad (6.1)$$

$$= \int_0^\infty C(\omega)\cos(\omega t + \Phi(\omega)) d\omega \qquad (6.2)$$

$$= \int_0^{\infty} D(\omega)\sin(\omega t + \theta(\omega))d\omega \tag{6.3}$$

$$= \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} E(\omega)e^{+i\,\omega t}d\omega \tag{6.4}$$

$$= \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(\omega)e^{+i\,\omega t}d\omega \tag{6.5}$$

$$= \int_{-\infty}^{+\infty} G(\nu)e^{+i\,2\,\pi\,\nu t}d\nu \tag{6.6}$$

Die Auflösung der Funktionen A bis G stellen das Hauptproblem der Fourieranalyse dar. Wohingegen die Funktionen $E(\omega)$, $F(\omega)$ und $G(\nu)$ als eigentliche Fouriertransformationen von $f(t)$ betrachtet werden können.

Hier gibt es keine allgemeingültig akzeptierte Konvention oder Nomenklatur der Schreibweise für die Hin- und Rücktransformation, und deshalb wollen wir uns hier auf eine Darstellung einigen und legen die Gleichung (6.6) als Grundlage für die weiteren Berechnungen fest. Die Funktionen $f(t)$ und $G(\nu)$ verallgemeinern wir und schreiben nun dafür $h(t)$ und $H(f)$. Es ist angebracht, $h(t)$ und $H(f)$ als unterschiedliche Repräsentationen der gleichen Funktion zu betrachten. Die symmetrische Fouriertransformation nach Waever wird dann wie folgt definiert:

Hintransformation: $$H(f) = \int_{-\infty}^{\infty} h(t)e^{2\pi ift}dt = FT^{-}\{h(t)\} \tag{6.7}$$

Rücktransformation: $$h(t) = \int_{-\infty}^{\infty} H(f)e^{-2\pi ift}df = FT^{+}\{H(f)\} \tag{6.8}$$

Der Zusammenhang zwischen f und ω bzw. $H(\omega)$ und $H(f)$ ist gegeben mit:

$$\omega = 2\pi f \qquad H(\omega) \equiv [H(f)]_{f=\omega/2\pi} \tag{6.9}$$

Wenn unter (6.7) und (6.8) t in Sekunden angegeben wird, dann sind f die Umdrehungen pro Sekunde oder besser: Hertz, der Einheit der Frequenz. Diese Gleichungen funktionieren auch mit anderen Einheiten: falls h eine Funktion der Position x in Metern ist, dann ist H die inverse Funktion, nämlich die der Wellenlänge, angegeben in Umdrehungen pro Meter.

Selbst wenn wir der Einfachheit halber mit der f-Konvention arbeiten, müssen doch hier und da ein paar 2π Faktoren in Zusammenhang mit diskret abgetasteten Werten beachtet werden. Diese werden zur rechten Zeit angeführt.

6.2 Sätze

Manchmal ist es hilfreich, ein Gefühl für den Zusammenhang zu bekommen, welche Effekte Operationen einer Funktion in einer Domäne haben und welche Wirkung diese auf ihre Transformierte ausübt. Deshalb seien hier noch ein paar Gesetze der Vollständigkeit halber aufgeführt.

(I) **Scallierung**: $h(at) \Leftrightarrow \frac{1}{|a|} H\left(\frac{f}{a}\right)$ (a = reale Konstante) (6.10)

$$\frac{1}{|a|} h\left(\frac{t}{a}\right) \Leftrightarrow H(af) \tag{6.11}$$

Eine Streckung der Zeitachse bewirkt eine Stauchung der Frequenzachse. Wird a<0 gewählt, so bewirkt das eine Umkehrung der Zeitachse und somit eine Umkehrung der Frequenzachse.

(II) **Verschiebung**: $h(t \pm t_0) \Leftrightarrow H(f) e^{\pm 2\pi i f t_0}$ (6.12)

$$h(t) e^{\pm 2\pi i f_0 t} \Leftrightarrow H(f \mp f_0) \tag{6.13}$$

Die Verschiebung in der Zeitdomäne führt zur Modulation in der Frequenzebene, und umgekehrt führt eine Modulation in der Zeitebene zur Verschiebung in der Frequenzebene.

Der Betrag der Fouriertransformierten einer verschobenen Zeitfunktion bleibt jedoch gleich. Die Verschiebung in der Zeitdomäne findet sich in der Phase der Polardarstellung wieder:

$$\tan(\phi) = \frac{\mathrm{Re}\{H(f)\}}{\mathrm{Im}\{H(f)\}} \tag{6.14}$$

(III) **Faltung / Convolution**: Die Fouriertransformation des Produktes zweier Funktionen ist proportional zur Faltung ihrer Transformationen:

$$g(t)h(t) \Leftrightarrow \{G(f) \otimes H(f)\} \tag{6.15}$$

Umgekehrt ist die Transformation der Faltung zweier Funktionen gleich dem Produkt der einzelnen Transformationen:

$$\{g(t) \otimes h(t)\} \Leftrightarrow G(f)H(f) \tag{6.16}$$

Das Produkt zweier Funktionen in der Zeitdomäne wird ausgedrückt mit $g \otimes h$ und definiert die Faltung – im Englischen auch Folding oder Convolution genannt – als:

$$g(t) \otimes h(t) \equiv \int_{=\infty}^{\infty} g(\tau) h(t-\tau) d\tau \qquad (6.17)$$

Die Faltung ist kommutativ (mit Hilfe der Substitution $\tau = t - \tau_2$), distributiv und assoziativ. Die Reihenfolge mehrerer Faltungen spielt keine Rolle.

(IV) **Flächen**: Die Fläche unter einer Funktion ist gleich dem Wert der Transformierten im Ursprung. Umgekehrt ist der Wert einer Funktion im Ursprung gleich der Fläche der Transformierten. Dies ist das Theorem von Parseval:

$$\int g(t) dt = G(0) \qquad \int G(f) df = g(0) \qquad (6.18)$$

Ein Spezialfall des Theorems behauptet, dass die Fläche unter der quadrierten Kurve in der Zeitdomäne mit der in der Frequenzdomäne gleich ist.

$$\int |g(t)|^2 dt = \int |G(f)|^2 df \qquad (6.19)$$

6.3 Fouriertransformation mit diskreten Werten

Normalerweise kennt man bei abgetasteten Werten einer Audioaufnahmen nicht deren kontinuierliche Funktion, sondern nur deren N diskrete Werte zu bestimmten, meist gleichverteilten Zeitpunkten. Die diskreten Werte der Funktion zu den Abtastzeitpunkten sind gegeben durch:

$$h_k \equiv h(t_k), \qquad t_k \equiv k \cdot \Delta t, \qquad \text{mit } k = 0,1,...,N-1 \qquad (6.20)$$

Der Kehrwert des Zeitintervalls Δ ist die Abtastrate. Falls Δ in Sekunden gemessen wird, so wird die Abtastrate in $1/s$ oder s^{-1} angegeben.

6.3.1 Abtasttheorem und Aliasing

Für jedes Abtastintervall Δ existiert eine kritische Nyquist Frequenz f_c. Sie ist gegeben durch:

$$f_c = \frac{1}{2\Delta} \qquad (6.21)$$

Das Nyquist Theorem sagt aus, dass eine Sinus-Frequenz mindestens zweimal während ihrer Periodenlänge abgetastet werden muss, nämlich zu den Zeitpunkten ihrer Maxima, wenn wir die Spektralkomponente dieser Kurve mit untersuchen wollen. Ein sogenannter Aliasing Effekt tritt auf, wenn wir dies nicht beachten. Frequenzen, welche außerhalb der kritischen Nyquist Frequenz liegen, werden in das aufgenommene Spektrum als falsche Frequenzen hineinkopiert und verfälschen das Ergebnis der Transformation.

Wir wissen aber, dass die Lautsprecher der Mobiltelephone bandpassbegrenzt sind. Sie sind spezifiziert, Frequenzen von ca. 400Hz bis 6000Hz ausgeben zu können. Mit der Angabe der maximalen Frequenz können wir die Mindestabtastrate bestimmen:

$$2 \cdot f_c = 2 \cdot 6000Hz = 12000s^{-1}.$$

Innerhalb Matlab können nur bestimmte Abtastraten eingestellt werden. Die nächst höhere ist mit 22050 angegeben. Wir nehmen also mindestens drei Abtastungen der höchsten vorkommenden Frequenz auf und sind damit auf sicherer Seite.

Es gibt noch einen weiteren Punkt, welcher in Zusammenhang mit dem Aliasing Effekt beachtet werden muss, und dies ist die Eigenschaft der Periodizität der Fouriertransformation. Die aufgenommenen Audiosignale sind nicht periodisch. Der Anfang und das Ende der Aufnahme besitzen meist unterschiedliche Größen, unter anderem durch Nebeneffekte wie Rauschen bedingt. Aus diesem Grund müssen die Aufnahmen so mit einer Fensterfunktion manipuliert werden, dass sie als periodisch fortsetzbar betrachtet werden können. Die Fensterfunktion von Tukey (6.22) stellt eine gute Wahl dar.

$$w(n) = \begin{cases} 1.0, & 0 \le |n| < \frac{N}{2}(1+r) \\ 0.5\left(1.0 + \cos\left(\pi \frac{n - \frac{N}{2}(1+r)}{N(1-r)}\right)\right), & \frac{N}{2}(1+r) \le |n| < N \end{cases} \qquad (6.22)$$

mit $n = 0,...,N-1$ und $0 \le r \le 1$

r steht für die Relation zwischen konstantem Verlauf und cos-Funktion.

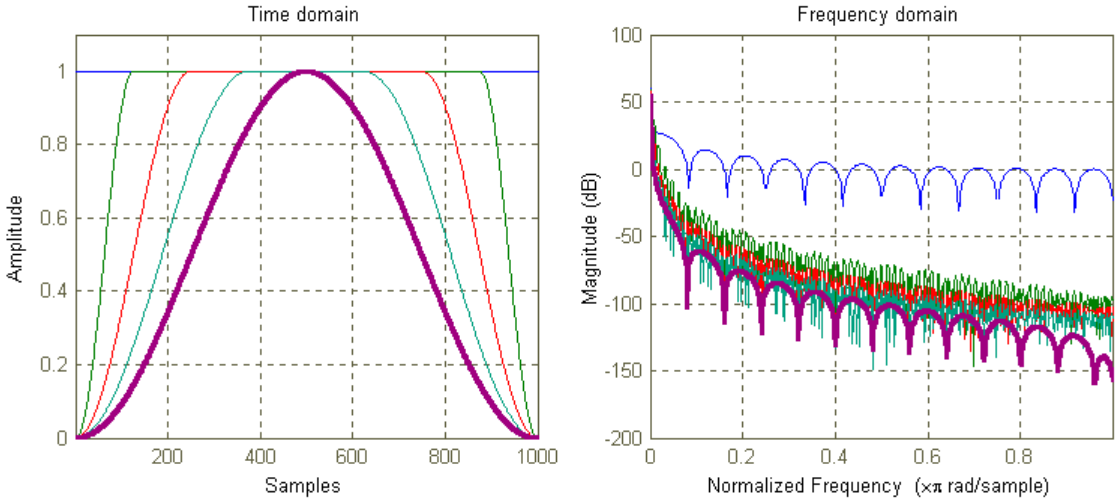


Abbildung 6.1: Die Tukeyfenster und ihre Fouriertransformierten in der „Power"-Darstellung sind mit $N = 1000$ und mit $r = 0$ (lila), $r = 0.25$, $r = 0.5$, $r = 0.75$ und $r = 1$ (blau) aufgetragen.

6.3.2 Diskrete Fouriertransformation

Mit N Werten in der Zeitdomäne lassen sich nicht mehr als N Werte in der Frequenzdomäne errechnen. Diese Werte f liegen im Bereich von $-f_c$ bis f_c. Daraus folgt:

$$f_n = \frac{n}{N\Delta}, \qquad \text{mit } n = -\frac{N}{2},...,\frac{N}{2} \tag{6.23}$$

Hier ist die höchste positive Frequenz gleich der höchsten negativen Frequenz. Diese beiden Extremwerte sind abhängig voneinander, da die Fouriertransformation annimmt, dass die aufgenommenen Werte periodische fortgeführt werden können. Wird die Aufnahme auf der Zeitachse um $N/2$ verschoben, dann nimmt die Transformation an, dass der Wert bei $n = N$ wieder gleich dem Wert bei $n = 0$ sei. Und daraus folgt, dass die Werte von $-f_c$ und f_c gleich sind.

Aus dieser Folgerung lässt sich der diskrete Teil der Transformation über die Integralsumme der kontinuierlichen Fouriertransformation aus (6.6) wie folgt annähern:

$$H(f_n) = \int_{-\infty}^{\infty} h(t) e^{2\pi i f_n t} dt \approx \sum_{k=0}^{N-1} h_k e^{2\pi i f_n t_k} \Delta = \Delta \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N} \tag{6.24}$$

Wobei $H(f_n) \approx \Delta H_n$ ist. Damit ergibt sich für die

Hintransformation:
$$H_n = \sum_{k=0}^{N-1} h_k e^{2\pi i k n / N}, \tag{6.25}$$

Rücktransformation:
$$h_k = \frac{1}{N} \sum_{n=0}^{N-1} H_n e^{-2\pi i k n / N} \tag{6.26}$$

6.4 Korrelation

Die **Kreuzkorrelation** ist ideal geeignet, um herauszufinden, ob eine gemessene Funktion $g(t)$ irgendetwas gemeinsam hat mit einer anderen gemessenen Funktion $h(t)$. Zwei völlig verschiedene und zueinander nicht zusammenhängende Funktionen werden eine relativ kleine Kreuzkorrelation für alle Werte von τ haben. Zwei Funktionen, die sich gut ähneln, z.B. zwei Aufnahmen des gleichen technischen Geräusches, werden eine relativ große Kreuzkorrelation für einige Werte von τ haben. Vergrößert sich t immer weiter, so tendiert die Kreuzkorrelation für alle Werte nach null, weil keine Abhängigkeit mehr vorhanden ist. Der höchste Wert, also der Peak der Korrelationsfunktion wird sich je nach Verschiebung von t auf gleiche Weise von null verschieben.

Die Kreuzkorrelation ist definiert mit:

$$Corr(f,g) = g(t) * h(t) = \int_{-\infty}^{\infty} g(\tau + t) h^*(\tau) d\tau \qquad (6.27)$$

$$= \int_{-\infty}^{\infty} g(\tau) h^*(\tau - t) d\tau \qquad (6.28)$$

oder diskret:

$$g_k * h_k = \frac{1}{N} \sum_{k=0}^{N-1} g(k+l) h^*(k) \qquad (6.29)$$

Der Stern $*$ bedeutet hierbei konjugiert komplex. Die Korrelation ist eine Funktion von t, dem Abstand oder Versatz der Funktion $g(t)$ zur Funktion $h(t)$. Das zugehörige Transformationspaar lautet:

$$g(t) * h(t) = G(f) H^*(f) \qquad (6.30)$$

Das heißt, dass die Multiplikation einer fouriertransformierten Funktion mit einer anderen fouriertransformierten und konjugierten Funktion, die Transformierte der Korrelation ergibt.

Die **Autokorrelation** ist ein Spezialfall der Kreuzkorrelation. Die Autokorrelationsfunktion ist ein Maß der Korrelation einer Funktion mit sich selbst und wird definiert durch:

$$Corr(g,g) = g(t) * g(t) = \int_{-\infty}^{\infty} g(\tau + t) g^*(\tau) d\tau \qquad (6.31)$$

oder diskret:

$$g_k * g_k = \frac{1}{N} \sum_{k=0}^{N-1} g(k+l) h^*(k) \qquad (6.32)$$

Das zugehörige Transformationspaar lautet somit:

$$g(t) * g(t) = |G(f)|^2 \qquad (6.33)$$

Es ist auch als Wiener-Khinchin Theorem bekannt. Es sagt aus, dass die Fouriertransformierte der Autokorrelationsfunktion gleich dem Informationsgehalt der Funktion $g(t)$ ist. Dieses Theorem wird auch als Power-Spektrum Dichte bezeichnet. Das Einseitige Power-Spektrum PSD (one-sided power spectral density) ist definiert durch:

$$P_h(f) = |H(f)|^2 + |H(-f)|^2 \qquad (0 \le f < \infty) \qquad (6.34)$$

Ist die Funktion $h(t)$ rein real, dann sind die beiden Terme in (6.34) gleich groß, und somit kann man für $P_h(f) = 2|H(f)|^2$ schreiben.

6.5 Analyse der Untersuchungen

In Tabelle 6.1 sind 10 unterschiedliche Aufnahmen von Musikstücken aus der Galerie eines Telephons graphische aufbereitet. Es wurden 5 Stücke ausgewählt, und diese wurden jeweils zweimal aufgenommen.

In der Diagonalen sind die Autokorrelationen der Aufnahmen zu sehen. Diese trennen die linke untere Hälfte der Tabelle mit den Funktionsplots der jeweiligen Kreuzkorrelationen von der rechten oberen Hälfte mit den zugehörigen Daten. Die Plots der Korrelationen wurden wegen der besseren Darstellung etwas verändert: Die linke Hälfte der Daten wurde mit der rechten Hälfte vertauscht. Die Daten aus der Tabelle geben Aufschluss über den Maximalausschlag der Kreuzkorrelation, der Gegenüberstellung zu den einzelnen Autokorrelationen sowie dem Unterschied der Gesamtkraft einer Aufnahmen zur anderen, also der Fläche unter der Kurve der Aufnahme. Als Beispiel seien hier die Ergebnisdaten von Kurve 1 zu 2 näher erläutert.

1. **4120 <.L> 154ms**: Diese Angaben bezeichnen die Höhe des maximalsten Ausschlags der Kreuzkorrelation und dessen Verschiebung. Der Maximalausschlag besitzt die Kraft 4120. Die Position dieses Wertes gibt Aufschluss über die Verschiebung beider Kurven gegeneinander. Hier ist Kurve 1 um 145ms linksverschoben gegenüber Kurve 2. Die x-Werte einer Korrelation stellen nur Summen der Funktionen zum Verschiebungszeitpunkt t dar (siehe (6.31)). Werden zwei Aufnahmen mit unterschiedlicher Lautstärke gemacht, werden auch die Werte der Korrelation niedriger ausfallen. Also kann noch keine Aussage über die Übereinstimmung beider Aufnahmen durch eine Kreuzkorrelation gemacht werden.

2. **d C(A): 0%**: Diese Angabe gibt den prozentualen Unterschied der Maximalwerte der Autokorrelationen der beiden zu untersuchenden Funktionen wieder. Je höher diese Angaben sind, desto schlechter müssen die Kurven zueinander passen. Hier haben beide Autokorrelationen den selben Maximalwert. Dieser Wert wird errechnet mit

$$Differenz = \left(\left| \frac{m_1}{m_2} - 1 \right| + \left| \frac{m_2}{m_1} - 1 \right| \right) * \frac{100}{2} \qquad \text{mit } m = \max(corr(...))$$

3. **d C(X-A): 17%**: Hier wird der prozentuale Unterschied zwischen den Maximalwerten der Autokorrelationen zur Kreuzkorrelation ($m1$) errechnet. Hier kann man bei weitem die größten Unterschiede zwischen gut korrelierenden Kurven und unkorrelierten Kurven erkennen. Bei unkorrelierten liegen die Werte zum Teil weit über der selbstgewählten 200%-Marke. Dieser Wert wird errechnet mit

$$Differenz = \left(\left| \frac{m_1}{mx} - 1 \right| + \left| \frac{mx}{m_1} - 1 \right| + \left| \frac{m_2}{mx} - 1 \right| + \left| \frac{mx}{m_2} - 1 \right| \right) * \frac{100}{4}$$

   Zunächst wurden die Unterschiede von einer Auto- zur Kreuzkorrelation und von der anderen Auto- zur Kreuzkorrelation gemessen. Dies ergab bei unkorrelierten Kurven meist mindestens einen hohen Wert; bei korrelierenden waren die Unterschiede weitgehend gering und die Werte niedrig. Aus diesen Gründen wurden beide Differenzen zu einer verschmolzen, welche etwa die gleichen Aussagen treffen kann.

4. **d P(.): 0%**: Bei dieser Angabe handelt es sich um den Unterschied der Flächen unter den zu vergleichenden Kurven. Beim Vergleich von Musikstücken befinden sich diese Prozentwerte weitgehend unter der 10%-Marke. Jedoch sollen nicht nur Musikstücke verglichen werden, sondern auch einfache Töne mit nur einer Frequenz. Diese Töne von anderen gleicher Frequenz aber länger anhaltender Welle zu unterscheiden ist mit der Aussage dieser Angabe etwas leichter. Hier prägen sich größere Unterschiede aus. Die Berechnung geschieht hier mit

$$Differenz = \left(\left| \frac{p_1}{p_2} - 1 \right| + \left| \frac{p_2}{p_1} - 1 \right| \right) * \frac{100}{2} \qquad \text{mit } p = \sum_{k=0}^{N-1} g(k)$$

5. **d P(A): 18%**: Diese Angabe zeigt den Unterschied zwischen den Mittelwerten der Funktionswerte beider Autokorrelationen. Je höher dieser Wert erscheint, desto größer korreliert eine Funktion mit sich selbst gegenüber der zu vergleichenden Kurve ihrer Autokorrelation. Bei länger anhaltenden gegenüber kürzeren Tönen mit reinen einfachem Frequenzen wirkt sich der Unterschied größer aus. Eine klare Aussage kann dieser Wert allein nicht liefern, sondern kann nur als Anhaltspunkt dienen. Er errechnet sich aus

$$Differenz = \left(\left| \frac{ma_1}{ma_2} - 1 \right| + \left| \frac{ma_2}{ma_1} - 1 \right| \right) * \frac{100}{2} \qquad \text{mit } ma = \frac{1}{N} \sum_{k=0}^{N-1} Corr_k[g(k)]$$

6. **d P(X-A): 15%**: Im Gegensatz zur oberen Angabe gibt dieser Wert Aufschluss des Unterschieds der einzelnen Kraft unter den Autokorrelationen gegenüber der Kreuzkorrelation von beiden Kurven. Die Größenordnungen dieser Angaben variieren je nach Vergleich der Daten. Werden Musikstücke verglichen, so müssten die Schwellwerte relativ hoch angesetzt werden, jedoch bei einfachen Tönen müssten sie niedriger gehalten werden. Vollständigkeitshalber sei hier noch kurz der Rechenweg:

$$Differenz = \left(\left| \frac{ma_1}{mx} - 1 \right| + \left| \frac{mx}{ma_1} - 1 \right| + \left| \frac{ma_2}{mx} - 1 \right| + \left| \frac{mx}{ma_2} - 1 \right| \right) * \frac{100}{4}$$

Summa Summarum lassen sich nur die Werte unter den Punkten 1.-4. weiterverwerten. Werden geeignete Schwellwerte hierfür angesetzt können einfache Vergleiche durchgeführt werden. Wird einer dieser Schwellwerte überschritten müsste ein Vergleich negativ ausfallen. Dies heißt jedoch nicht, dass sich die Aufnahmen dann in keinster Weise mehr ähneln. Die Kurven passen dann nur nicht mehr „perfekt“ zueinander. Nur über zusätzliche Mathematik oder weitere Zusatztests können Verbesserungen erzielt werden.

Um möglichst gute Ergebnisse bei der Aufnahme zu erzielen, musste der Schrank, in dem das Testgerät steht, mit Dämmmaterial ausgekleidet werden. Unter dieser Voraussetzung ließen sich das Rauschen und die Geräuscheinstreuungen von außen sowie der Grad der Übersteuerung des Signals erheblich reduzieren. Es konnten dadurch bessere Aufnahmen erzielt werden.

Sound-waves & Correlations	1	2	3	4	5	6	7	8	9	10
1		4120 <.L> 154ms d C(A): 0% d C(X-A): 17% d P(.): 0% d P(A): 18% d P(X-A): 15%	215 <.L> 1931ms d C(A): 23% d C(X-A): 1272% d P(.): 27% d P(A): 142% d P(X-A): 199%	219 <.L> 4957ms d C(A): 23% d C(X-A): 1250% d P(.): 26% d P(A): 65% d P(X-A): 153%	99 <.L> 7046ms d C(A): 57% d C(X-A): 1942% d P(.): 4% d P(A): 71% d P(X-A): 287%	125 <.R> 6479ms d C(A): 50% d C(X-A): 1565% d P(.): 0% d P(A): 63% d P(X-A): 198%	98 <.R> 5756ms d C(A): 211% d C(X-A): 6753% d P(.): 83% d P(A): 497% d P(X-A): 1216%	113 <.L> 9802ms d C(A): 211% d C(X-A): 5860% d P(.): 83% d P(A): 504% d P(X-A): 824%	110 <.R> 6863ms d C(A): 167% d C(X-A): 1405% d P(.): 29% d P(A): 46% d P(X-A): 209%	121 <.L> 8768ms d C(A): 161% d C(X-A): 1293% d P(.): 28% d P(A): 59% d P(X-A): 159%
2			194 <.L> 4831ms d C(A): 23% d C(X-A): 1415% d P(.): 26% d P(A): 113% d P(X-A): 262%	186 <.R> 5346ms d C(A): 22% d C(X-A): 1473% d P(.): 26% d P(A): 45% d P(X-A): 173%	104 <.R> 8673ms d C(A): 58% d C(X-A): 1847% d P(.): 4% d P(A): 50% d P(X-A): 335%	126 <.R> 7121ms d C(A): 51% d C(X-A): 1561% d P(.): 1% d P(A): 43% d P(X-A): 251%	122 <.R> 6921ms d C(A): 210% d C(X-A): 5405% d P(.): 82% d P(A): 414% d P(X-A): 1098%	89 <.R> 2275ms d C(A): 211% d C(X-A): 7470% d P(.): 83% d P(A): 420% d P(X-A): 1370%	100 <.L> 9730ms d C(A): 168% d C(X-A): 1549% d P(.): 30% d P(A): 27% d P(X-A): 259%	85 <.L> 2206ms d C(A): 162% d C(X-A): 1843% d P(.): 28% d P(A): 39% d P(X-A): 319%
3				4673 <.R> 58ms d C(A): 0% d C(X-A): 27% d P(.): 0% d P(A): 56% d P(X-A): 29%	234 <.L> 4531ms d C(A): 85% d C(X-A): 949% d P(.): 31% d P(A): 51% d P(X-A): 118%	284 <.R> 991ms d C(A): 78% d C(X-A): 799% d P(.): 27% d P(A): 59% d P(X-A): 59%	500 <.L> 6397ms d C(A): 163% d C(X-A): 1383% d P(.): 51% d P(A): 143% d P(X-A): 276%	884 <.L> 933ms d C(A): 163% d C(X-A): 779% d P(.): 51% d P(A): 146% d P(X-A): 71%	330 <.L> 9369ms d C(A): 216% d C(X-A): 557% d P(.): 58% d P(A): 76% d P(X-A): 59%	486 <.L> 442ms d C(A): 209% d C(X-A): 375% d P(.): 57% d P(A): 63% d P(X-A): 30%
4					446 <.R> 439ms d C(A): 85% d C(X-A): 493% d P(.): 30% d P(A): 5% d P(X-A): 19%	379 <.R> 3978ms d C(A): 77% d C(X-A): 593% d P(.): 27% d P(A): 2% d P(X-A): 21%	733 <.L> 6468ms d C(A): 163% d C(X-A): 941% d P(.): 52% d P(A): 263% d P(X-A): 117%	700 <.R> 8540ms d C(A): 163% d C(X-A): 986% d P(.): 52% d P(A): 267% d P(X-A): 126%	397 <.R> 3144ms d C(A): 216% d C(X-A): 459% d P(.): 58% d P(A): 17% d P(X-A): 16%	311 <.L> 1385ms d C(A): 209% d C(X-A): 594% d P(.): 56% d P(A): 5% d P(X-A): 37%
5						1670 <.R> 277ms d C(A): 6% d C(X-A): 58% d P(.): 4% d P(A): 7% d P(X-A): 3%	418 <.L> 9441ms d C(A): 377% d C(X-A): 1457% d P(.): 88% d P(A): 249% d P(X-A): 112%	278 <.R> 7023ms d C(A): 377% d C(X-A): 2198% d P(.): 88% d P(A): 253% d P(X-A): 303%	396 <.R> 2936ms d C(A): 81% d C(X-A): 252% d P(.): 25% d P(A): 22% d P(X-A): 17%	266 <.R> 4511ms d C(A): 77% d C(X-A): 388% d P(.): 24% d P(A): 10% d P(X-A): 26%
6							366 <.R> 3157ms d C(A): 354% d C(X-A): 1676% d P(.): 83% d P(A): 269% d P(X-A): 124%	328 <.R> 3079ms d C(A): 354% d C(X-A): 1872% d P(.): 83% d P(A): 273% d P(X-A): 197%	377 <.R> 291ms d C(A): 89% d C(X-A): 277% d P(.): 29% d P(A): 15% d P(X-A): 15%	347 <.L> 107ms d C(A): 85% d C(X-A): 306% d P(.): 28% d P(A): 3% d P(X-A): 21%
7								19827 <.L> 653ms d C(A): 0% d C(X-A): 9% d P(.): 0% d P(A): 1% d P(X-A): 7%	247 <.L> 4232ms d C(A): 801% d C(X-A): 2318% d P(.): 125% d P(A): 313% d P(X-A): 265%	250 <.R> 2310ms d C(A): 777% d C(X-A): 2295% d P(.): 122% d P(A): 278% d P(X-A): 327%
8									382 <.L> 6514ms d C(A): 802% d C(X-A): 1496% d P(.): 125% d P(A): 318% d P(X-A): 197%	450 <.R> 1751ms d C(A): 778% d C(X-A): 1270% d P(.): 122% d P(A): 282% d P(X-A): 125%
9										376 <.R> 9737ms d C(A): 3% d C(X-A): 167% d P(.): 1% d P(A): 11% d P(X-A): 7%
10										

Tabelle 6.1: Gegenüberstellung von ausgewählten Musikstücken

Tone-waves & Correlations	1: 880Hz – 800ms	2: 880Hz – 1200ms	3: 1245Hz – 900ms	4: 1397Hz – 900ms	5: 1397Hz – 950ms	6: 1397Hz – 1000ms	7: 1397Hz – 1200ms	8: 1860Hz – 900ms (1)	9: 1860Hz – 900ms (2)	10: 2489Hz – 900ms
1		623 <.R> 5ms d C(A): 59% d C(X-A): 35% d P(.): 41% d P(A): 106% d P(X-A): 48%	25 <.R> 202ms d C(A): 22% d C(X-A): 1185% d P(.): 9% d P(A): 13% d P(X-A): 1171%	17 <.R> 54ms d C(A): 6% d C(X-A): 1887% d P(.): 0% d P(A): 17% d P(X-A): 2040%	21 <.L> 163ms d C(A): 6% d C(X-A): 1505% d P(.): 1% d P(A): 6% d P(X-A): 1648%	21 <.L> 71ms d C(A): 1% d C(X-A): 1576% d P(.): 6% d P(A): 25% d P(X-A): 1815%	22 <.R> 617ms d C(A): 23% d C(X-A): 1649% d P(.): 21% d P(A): 58% d P(X-A): 2019%	21 <.L> 412ms d C(A): 300% d C(X-A): 5525% d P(.): 94% d P(A): 382% d P(X-A): 6823%	27 <.R> 456ms d C(A): 299% d C(X-A): 4312% d P(.): 94% d P(A): 379% d P(X-A): 5333%	20 <.R> 581ms d C(A): 247% d C(X-A): 4952% d P(.): 83% d P(A): 314% d P(X-A): 5663%
2		31 <.R> 339ms d C(A): 85% d C(X-A): 1331% d P(.): 51% d P(A): 125% d P(X-A): 1582%	22 <.R> 518ms d C(A): 66% d C(X-A): 2021% d P(.): 41% d P(A): 83% d P(X-A): 2263%	37 <.L> 370ms d C(A): 65% d C(X-A): 1198% d P(.): 40% d P(A): 98% d P(X-A): 1575%	31 <.R> 583ms d C(A): 58% d C(X-A): 1458% d P(.): 35% d P(A): 72% d P(X-A): 1719%	36 <.R> 154ms d C(A): 33% d C(X-A): 1375% d P(.): 19% d P(A): 38% d P(X-A): 2184%	25 <.L> 213ms d C(A): 163% d C(X-A): 5184% d P(.): 45% d P(A): 138% d P(X-A): 5499%	46 <.R> 488ms d C(A): 162% d C(X-A): 2802% d P(.): 45% d P(A): 137% d P(X-A): 3908%	28 <.R> 524ms d C(A): 130% d C(X-A): 4092% d P(.): 37% d P(A): 109% d P(X-A): 4653%
3			21 <.R> 58ms d C(A): 16% d C(X-A): 1379% d P(.): 8% d P(A): 29% d P(X-A): 1825%	25 <.L> 61ms d C(A): 16% d C(X-A): 1127% d P(.): 10% d P(A): 18% d P(X-A): 1307%	26 <.L> 188ms d C(A): 23% d C(X-A): 1140% d P(.): 14% d P(A): 38% d P(X-A): 1370%	25 <.L> 740ms d C(A): 46% d C(X-A): 1331% d P(.): 30% d P(A): 73% d P(X-A): 1369%	27 <.L> 616ms d C(A): 377% d C(X-A): 4252% d P(.): 106% d P(A): 434% d P(X-A): 6105%	26 <.L> 161ms d C(A): 376% d C(X-A): 4443% d P(.): 106% d P(A): 432% d P(X-A): 5360%	32 <.L> 184ms d C(A): 312% d C(X-A): 3060% d P(.): 95% d P(A): 358% d P(X-A): 4685%
4				550 <.L> 3ms d C(A): 0% d C(X-A): 11% d P(.): 1% d P(A): 11% d P(X-A): 5%	576 <.L> 42ms d C(A): 7% d C(X-A): 10% d P(.): 6% d P(A): 8% d P(X-A): 4%	581 <.L> 115ms d C(A): 29% d C(X-A): 21% d P(.): 21% d P(A): 40% d P(X-A): 19%	16 <.R> 115ms d C(A): 320% d C(X-A): 7441% d P(.): 94% d P(A): 322% d P(X-A): 8070%	18 <.L> 510ms d C(A): 319% d C(X-A): 6498% d P(.): 94% d P(A): 320% d P(X-A): 8021%	16 <.R> 380ms d C(A): 264% d C(X-A): 6096% d P(.): 83% d P(A): 264% d P(X-A): 7948%
5					548 <.L> 39ms d C(A): 7% d C(X-A): 15% d P(.): 5% d P(A): 19% d P(X-A): 10%	556 <.L> 251ms d C(A): 29% d C(X-A): 25% d P(.): 20% d P(A): 51% d P(X-A): 25%	18 <.L> 562ms d C(A): 320% d C(X-A): 6345% d P(.): 92% d P(A): 360% d P(X-A): 7525%	24 <.L> 183ms d C(A): 318% d C(X-A): 4821% d P(.): 93% d P(A): 358% d P(X-A): 5667%	21 <.R> 716ms d C(A): 263% d C(X-A): 4649% d P(.): 82% d P(A): 296% d P(X-A): 5837%
6						610 <.L> 59ms d C(A): 22% d C(X-A): 19% d P(.): 15% d P(A): 31% d P(X-A): 15%	22 <.L> 425ms d C(A): 298% d C(X-A): 5354% d P(.): 86% d P(A): 294% d P(X-A): 6867%	21 <.R> 443ms d C(A): 297% d C(X-A): 5611% d P(.): 87% d P(A): 293% d P(X-A): 6046%	18 <.R> 263ms d C(A): 245% d C(X-A): 5563% d P(.): 76% d P(A): 241% d P(X-A): 7338%
7							23 <.R> 102ms d C(A): 235% d C(X-A): 5295% d P(.): 67% d P(A): 213% d P(X-A): 7026%	29 <.L> 724ms d C(A): 234% d C(X-A): 4239% d P(.): 67% d P(A): 211% d P(X-A): 6120%	21 <.L> 408ms d C(A): 192% d C(X-A): 5010% d P(.): 58% d P(A): 172% d P(X-A): 5609%
8								3956 <.R> 10ms d C(A): 0% d C(X-A): 2% d P(.): 0% d P(A): 1% d P(X-A): 0%	21 <.L> 446ms d C(A): 18% d C(X-A): 8934% d P(.): 8% d P(A): 19% d P(X-A): 10258%
9									24 <.L> 739ms d C(A): 18% d C(X-A): 7670% d P(.): 8% d P(A): 18% d P(X-A): 9689%
10									

Tabelle 6.2: Gegenüberstellung von ausgewählten Tönen von einer Frequenz und variierenden Längen

6.6 SNDCMP [*time*]

Im Unterpunkt 6.6.1 wird die Funktion Sound das Austere-Systems kurz erläutert und im Unterkapitel 6.6.2 wird die angewandte Mathematik des Vergleichs erläutert.

6.6.1 die Sound Funktion

Das Kommando „sndcmp“ nimmt einen Ton über den Standartmikrophoneingang auf und vergleicht diesen mit dem Referenzton aus der Datenbank. Der Parameter „time“ kann optional in Sekunden angegeben werden. Der Standartwert beträgt eine Sekunde. Dem Matlab Skript, in welchem die Hauptaufgabe der Aufnahme und des Vergleichs letztlich geschieht, werden drei Parameter übergeben. In den ersten beiden Parametern werden die Dateinamen übergeben. „out_fn“ bezeichnet den Namen, unter dem die Aufnahme (oder auch das Resultat) abzulegen ist, und „ref_fn“ bezeichnet die Datei in der der Referenzton aus der Datenbank abgespeichert wurde. Im Parameter „tm“ wird die aufzunehmende Zeit übergeben. Der vorgeschaltete Block „extract time [s]“ versucht den optionalen Parameter „time“ des Befehls zu extrahieren.

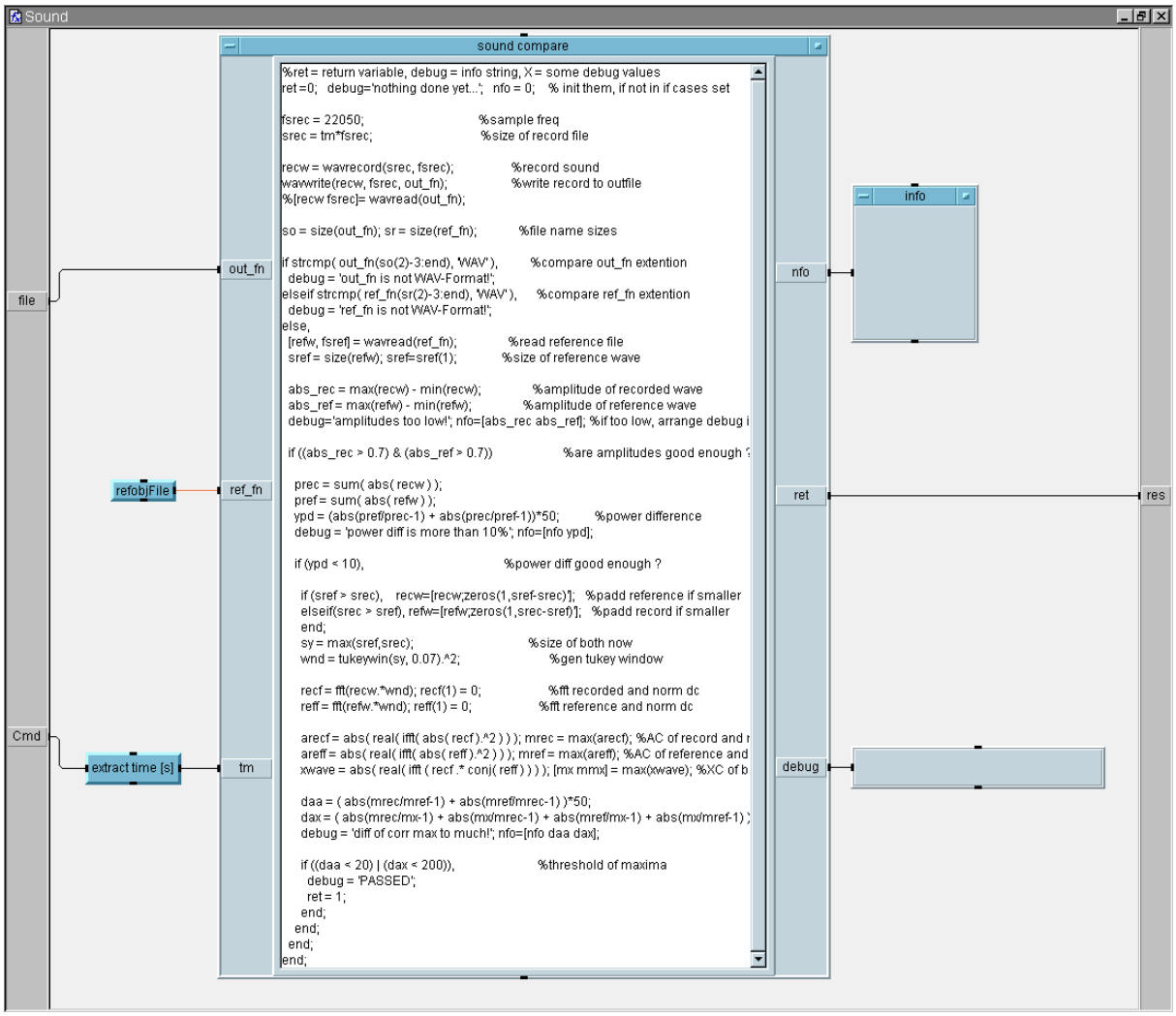


Abbildung 6.2: Die Implementierung des Kommandos sndcmp.

Die Ausgänge des Skripts sind: „ret“, „nfo“ und „debug“. „ret“ gibt das Resultat im Ganzzahlformat zurück: eine Eins, wenn der Vergleich positiv ausgefallen ist und eine Null, wenn der Vergleich negativ ausgefallen ist. Das Informationsfenster „Info“ wird mit den Vergleichswerten des Skripts über den Ausgang „nfo“ befüllt. Im Ausgang „debug“ erscheint eine kurze textuelle Meldung über den Ausgang des Vergleichs. Die letzteren zwei Ausgänge und ihre Werte werden im nächsten Unterkapitel mit dem Code beschrieben.

6.6.2 Das Matlab Skript “sound compare”

Dieses Matlab Skript nimmt den Ton, welcher zu vergleichen ist auf und führt den Vergleich durch. Nachfolgend wird die angewandte Algorithmus näher erklärt.

Die Ausgänge werden zu allererst konfiguriert, weil gegebenenfalls uninitialisierte Ausgänge am Ende des Skripts mit einem Fehler bei der Ausführung enden. Die Variable „ret“ wird auf den Wert **0** für „failed“ gesetzt. Die Debugausgabe wird sich je nach Ausführungsweite ändert, ebenso die Variable „nfo“.

```
ret = 0;   debug = 'nothing done yet...';   nfo = 0;
```

Vor Beginn jeglicher Aktivität sollte geprüft werden, ob die Referenz eine Audiodatei ist und ob der Dateiname zur Speicherung der Aufnahme ebenfalls eine Audiodatei sein soll. Hin und wieder kann es vorkommen, dass Testskripte umgearbeitet werden und dass eine alte Referenz mit anderem Format als „wav“ mit einer Aufnahme verglichen werden soll. Ein einfacher Vergleich der Dateinamensendung reicht hier völlig aus.

```
so = size(out_fn); sr = size(ref_fn);        %file name sizes
if strcmp( out_fn(so(2)-3:end), 'WAV' ),     %compare out_fn extention
  debug = 'out_fn is not WAV-Format!';
elseif strcmp( ref_fn(sr(2)-3:end), 'WAV' ), %compare ref_fn extention
  debug = 'ref_fn is not WAV-Format!';
else,
```

Vor Aufnahmebeginn wird die Abtastrate, sowie die Gesamtlänge der Aufnahme festgelegt.

```
   fsrec = 22050;                            %sample rate
   srec = tm*fsrec;                          %size of recording
```

Anschließend kann die Aufnahme beginnen und in der angegebenen Datei „`out_fn`“ abgespeichert werden.

```
   recw = wavrecord(srec, fsrec);            %recording
   wavwrite(recw, fsrec, out_fn);            %writing record
```

Nun wird die Referenz aus der Datei geladen und ihre Größe bestimmt.

```
   [refw, fsref] = wavread(ref_fn);          %read reference file
   sref = size(refw); sref=sref(1);          %size of reference wave
```

Sicherheitshalber werden die Höhen der Amplituden beider Audiodaten bestimmt. Liegen diese unter einem bestimmten Minimum, dann liegt zu wenig Kraft unter der Kurve. Ein anderer Grund liegt in der Höhe des mit aufgenommenen Rauschens. Dieses lässt sich leider nur bis zu einem bestimmten Grad unterdrücken und würde bei sehr leisen Aufnahmen diese übertönen.

```
abs_rec = max(recw) - min(recw);          %amplitude of recordedwave

abs_ref = max(refw) - min(refw);          %amplitude of reference wave

debug='amplitudes too low!'; nfo=[abs_rec abs_ref];
```

Falls diese Amplitudenhöhen zu niedrig sind, wird der weitere Vergleich hier abgebrochen. Die alte Debuginformation wurde zuvor von der neuen überschrieben.

```
if ((abs_rec > 0.7) & (abs_ref > 0.7))    %are amplitudes good enough?
```

Als nächstes werden die Kräfte unter den Kurven gegeneinander verglichen. Dieses berechnet sich im diskreten Fall aus der Summe der Abtastwerte. Im kontinuierlichen Fall aus dem Integral der Kurve. Im Abschnitt 6.5 im 4. Punkt wurde auf den Vergleich beider Werte eingegangen und wird nachfolgend berechnet unter „`ypd`“.

```
prec = sum( abs( recw ) );

pref = sum( abs( refw ) );

ypd = (abs(pref/prec-1) + abs(prec/pref-1))*50;    %diff in power

debug = 'power diff is more than 10%'; nfo=[nfo ypd];
```

Falls der eingestellte Schwellwert von 10% unterschritten wird, so wird der weitere Vergleich hier abgebrochen. Die Debuginformation wurde zuvor auf den negativen Ausgang des Vergleichs angepasst.

```
if (ypd < 10),                            %power diff good enough ?
```

Nachdem der Vergleich bis hier positiv verlief, müssen nun die gegebenenfalls unterschiedlichen Längen der Kurven angepasst und gleichgroß gemacht werden. Dies geschieht mittels Auffüllung mit Nullen, neudeutsch auch Zeropadding genannt.

```
if (sref > srec),    recw=[recw;zeros(1,sref-srec)'];   %padd ref

elseif(srec > sref), refw=[refw;zeros(1,srec-sref)'];   %padd record

end;
```

Die Größe der Aufnahmen ist nun der Maximalwert beider Größen der Aufnahmen.

```
sy = max(sref,srec);                      %size of both now
```

Um beide Aufnahmen periodisch aussehen zu lassen, wird nun ein quadriertes Tukey Fenster mit $g(t) * g(t) = |G,f)|^2$ generiert.

```
wnd = tukeywin(sy, 0.07).^2;              %gen tukey window
```

Anschließend können beide Kurven fouriertransformiert werden. Der Gleichanteil in den Kurven wird hier unterschlagen und geht nicht weiter in spätere Berechnungen ein.

```
recf = fft(recw.*wnd); recf(1) = 0;   %fft recorded and norm dc

reff = fft(refw.*wnd); reff(1) = 0;   %fft reference and norm dc
```

Nach der Transformation können die Korrelationen berechnet und deren Maximalwerte mit vorgegebenen Schwellwerten verglichen werden.

```
arecf = abs( real( ifft( abs( recf ).^2 ) ) );    %autocorr of rec
mrec = max(arecf);                                % max(arecf)
areff = abs( real( ifft( abs( reff ).^2 ) ) );    %autocorr of ref
mref = max(areff);                                %max(areff)
xwave = abs( real( ifft ( recf .* conj( reff ) ) ) );   %crosscorr
[mx mmx] = max(xwave);                            %max(xc)
```

Die Hintergründe der nachfolgenden Vergleiche sind im Kapitel 6.5 in den Unterpunkten 2 und 3 näher beschrieben.

```
daa = ( abs(mrec/mref-1) + abs(mref/mrec-1) )*50;
dax = ( abs(mrec/mx-1) + abs(mx/mrec-1) +
        abs(mref/mx-1) + abs(mx/mref-1) )*25;
debug = 'diff of corr max to much!'; nfo=[nfo daa dax];
```

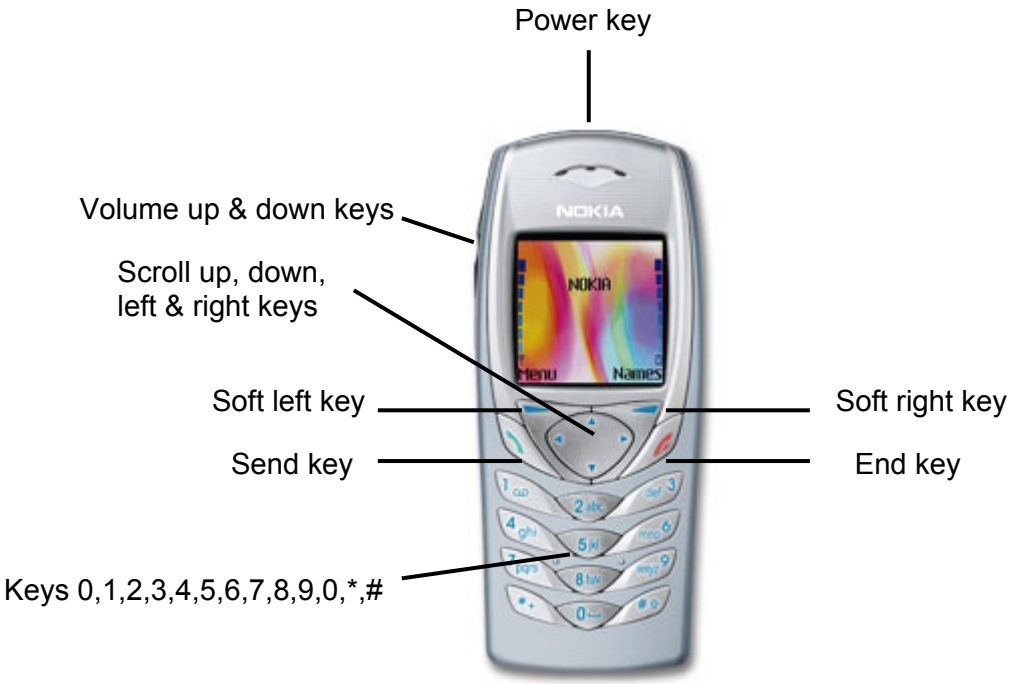
Dieser Vergleich ist dann der letzte Vergleich. Wenn dieser Bestanden ist, dann wird das Vergleichsresultat auf **1** gesetzt, welches ein „passed“ bedeutet.

```
if ((daa < 20) | (dax < 200)),          %compare with threshold
 ret = 1;
end;
  end;
    end;
      end;
```

Bei noch genauerer und intensiverer Auseinandersetzung mit diesem Thema ließe sich noch einiges verfeinern und erweitern. Dies würde jedoch den Rahmen dieser Diplomarbeit bei weitem sprengen. Dieser Lösungsansatz ist für das Testsystem völlig ausreichend. Die Berechnungszeit hält sich in Grenzen und gut aufgenommene Töne können einfach verglichen werden.

7 TASTENBELEGUNG EINES MOBILTELEPHONS

Beispielhaft sei hier die Tastenbelegung eines Mobilfunktelephons vom Typ Nokia 6100 dargestellt:



8 DANKSAGUNG

Für die gute Hilfe und Unterstützung während meiner Arbeit!

- Detlev Albold
- Michael Roos
- Helmut Malz
- Katrin Mertens
- Alexander Ross
- Roland Heinemann
- Andreas Heinemann
- Tilla Scholz
- Ingo Zohren

9 LITERATURNACHWEIS

- Benutzerhandbuch und Technische Referenz von VEE Pro Version 6.1, Copyright Agilent Technology Inc., 1999.
- DisplayInspect User's Manual, Revision 1.3, Copyright Cognex Co., 1998.
- Elfriede Dustin, Jeff Rashka, John Paul: Software automatisch Testen. Verfahren, Handhabung und Leistung. Berlin, Heidelberg, New York: Springer, 2000.
- Sun Microsystems' HomePage: http://www.sun.com.
- Nokia Intranet (stellvertretend: http://www.nokia.de, http://www.nokia.com).
- Eberhard Schöneburg, Nikolaus Hansen, Andreas Gawelczyk: Neuronale Netzwerke: Einführung, Überblick und Anwendungsmöglichkeiten. Haar: Markt u. Technik, 1990.
- Yoh-Han Pao: Adaptive Pattern Recognition and Neural Networks. Reading, Addison-Wesley, 1989.
- Donald R. Tveter: Backpropagator's Review, http://www.dontveter.com/bpr/bpr.html.
- W. Press, B. Flannery, S. Teukolsky, W. Vetterling: Numerical Recipes: The Art of Scientific Computing. New York: Press Syndicate of the University of Campridge, 1989.
- D. Champeney: Fourier Transforms and their Physical Application. London, New York: Academic Press, 1973.
- T. Butz: Fouriertransformation für Fußgänger. Stuttgart, Leipzig: B.G.Teubner, 1998.
- Mathworks Online Helpdesk, http://www.mathworks.com.

Ehrenwörtliche Erklärung

Hiermit erkläre ich, Sven Mertens, geboren am 27.12.1972 in Weimar/Thüringen, ehrenwörtlich,

(1) dass ich meine Diplomarbeit mit dem Titel:

„Erweiterung des automatischen Testsystems Austere„

im Betrieb Nokia unter Anleitung von Detlev Albold selbständig und ohne fremde Hilfe angefertigt habe und keine anderen als in der Abhandlung angeführten Hilfen benutzt habe;

(2) dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Konstanz, 28.02.2003 Sven Mertens